Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Petelin

Globoke nevronske mreže in matrična faktorizacija

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Igor Kononenko

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Globoke nevronske mreže so usmerjene večnivojske nevronske mreže z mnogo nivoji skritih nevronov (več kot tri, lahko tudi nekaj deset nivojev), ki omogočajo z dodatnimi skritimi nivoji tvorjenje novih atributov in lahko dosegajo v praksi odlične napovedne točnosti. Vendar je potrebno za vsak problem posebej določiti strukturo mreže in parametre učenja ter izbrati ustrezno inicializacijo uteži skritim nevronom, ker je sicer učenje iz naključnega začetnega stanja prekompleksno. Matrična faktorizacija je postopek, kjer se skuša zmanjšati predstavitev vhodnih podatkov (velike vhodne matrike) tako, da se jo aproksimira s produktom dveh manjših matrik. Matrična faktorizacije sodi na področje nenadzorovanega učenja, ki se lahko interpretira kot razvrščanje (clustering) in ki se lahko uporabi za inicializacijo uteži pri globokih nevronskih mrežah. Naloga je raziskati področje inicializacije globokih nevronskih mrež s pomočjo matrične faktorizacije, zatem pa implementirati in pretestirati nekaj arhitektur globoke nevronske mreže z nekaj različnimi matričnimi faktorizacijami in primerjati razultate na različnih podatkovnih bazah.

Zahvaljujem se prof. dr. Igorju Kononenku za usmerjanje, pomoč in strokovne nasvete pri izdelavi diplomske naloge. Zahvaljujem se tudi staršem in drugim sorodnikom, ki so me podpirali pri izobraževanju, ter prijateljem, ki so mi v času študija delali družbo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
CA	classification accuracy	klasifikacijska točnost
MAE	mean absolute error	srednja absolutna napaka
NMF	non-negative matrix factorization	nenegativna matrična faktorizacija
AA	archetypal analysis	analiza arhetipov
ANN	artificial neural network	umetna nevronska mreža
PCA	principal component analysis	metoda glavnih komponent
SVD	singular value decomposition	singularni razcep
FA	factor analysis	faktorska analiza

Povzetek

Naslov: Globoke nevronske mreže in matrična faktorizacija

Avtor: Gašper Petelin

V diplomski nalogi preizkusimo novo metodo inicializacije uteži nevronskih mrež, kjer vhodno matriko podatkov faktoriziramo v več manjših matrik, v katerih je zgoščena predstavitev podatkov. Iz teh matrik zgradimo in naučimo več enonivojskih nevronskih mrež, ki preslikajo podatke iz ene zgoščene predstavitve, dobljene z faktorizacijo, v drugo predstavitev, na koncu pa dodamo še nevronsko mrežo, ki preslika podatke iz zadnje zgoščene predstavitve podatkov v pravilne razrede pri klasifikaciji oziroma v pravilno vrednost pri regresiji. Vse te naučene enonivojske mreže nato združimo v ciljno nevronsko mrežo in jo še dodatno učimo.

Ta postopek je običajno boljši pri inicializaciji globokih nevronskih mrež, kjer pri naključni inicializaciji pogosto prihaja do zelo počasnega učenja. Za delovanje in primerjavo inicializacije so uporabljeni podatki MNIST za klasifikacijo in Jester jokes za regresijo.

Ključne besede: globoke nevronske mreže, klasifikacijska točnost, matrična faktorizacija, analiza arhetipov, inicializacija uteži.

Abstract

Title: Deep neural networks and matrix factorization

Author: Gašper Petelin

This thesis proposes a new data-driven method for neural network weight initialization, where input data matrix is first factorized into multiple smaller matrices, each containing a summarized version of original data. Multiple shallow neural networks are then trained using acquired smaller matrices to learn simple functions, mapping one summarized data matrix into another, usually smaller matrix. One last shallow neural network is added to map the last summarized data matrix into their respective class labels if we are trying to classify data into multiple classes. On the other hand, if we are dealing with regression problem, the last neural network represents a simple mapping from summarized data into a single real value. All shallow neural networks are then combined into one deep network and additionally trained as a single neural network.

The proposed method usually works better for deep neural networks, where random initialization often overfits or learns very slowly.

To evaluate and compare the proposed method with other initialization methods, two datasets were used. The MNIST dataset was used to test classification accuracy and the Jester jokes dataset was used to predict ratings for individual jokes.

Keywords: deep neural networks, classification accuracy, matrix factorization, archetypal analysis, weight initialization.

Poglavje 1

Uvod

Umetne nevronske mreže so algoritmi na področju strojnega učenja, ki trenutno na določenih področjih dosegajo vrhunske rezultate. Uporabljene so za vrsto različnih problemov, kot so prepoznavanje govora [8], klasifikacija slik [14] in analiza časovnih vrst [19]. Od samega začetka razvoja nevronske mreže niso bile tako uspešne, saj so se sprva srečevale z različnimi problemi, kot so pomankanje metod za nastavljanja uteži in kasneje s težavo izginjanja gradienta. Običajno se uteži v nevronskih mrežah nastavijo naključno, vendar če so te nevronske mreže precej globoke, to največkrat pripelje do izgube gradienta.

V diplomski nalogi predstavimo novo metodo inicializacije uteži nevronskih mrež, ki je nekoliko bolj časovno zahtevna, vendar za globoke nevronske mreže običajno dosega boljše rezultate kot naključne inicializacije.

V prvem razdelku 2 so najprej na kratko predstavljene nevronske mreže in algoritem za vzratno razširjanje napake. Opisane so tudi določene težave, s katerimi se nevronske mreže srečujejo, kot so izguba gradienta in preveliko prileganje, ter nekateri postopki, ki jih uporabljamo, da se temu izognemo.

Naslednji razdelek 3 opisuje, kaj je matrična faktorizacija, njeno uporabo ter kateri tipi matrične faktorizacije sploh obstajajo in kaj so njihove prednosti in slabosti.

Razdelek o inicializaciji uteži 4 opisuje dve najbolj uporabljeni metodi

inicializacije, ki naključno nastavita vrednosti uteži in težave, s katerimi se srečujeta. Opisana je tudi predlagana metoda, kjer uteži inicializiramo s pomočjo matrične faktorizacije in učenja enonivojskih nevronskih mrež.

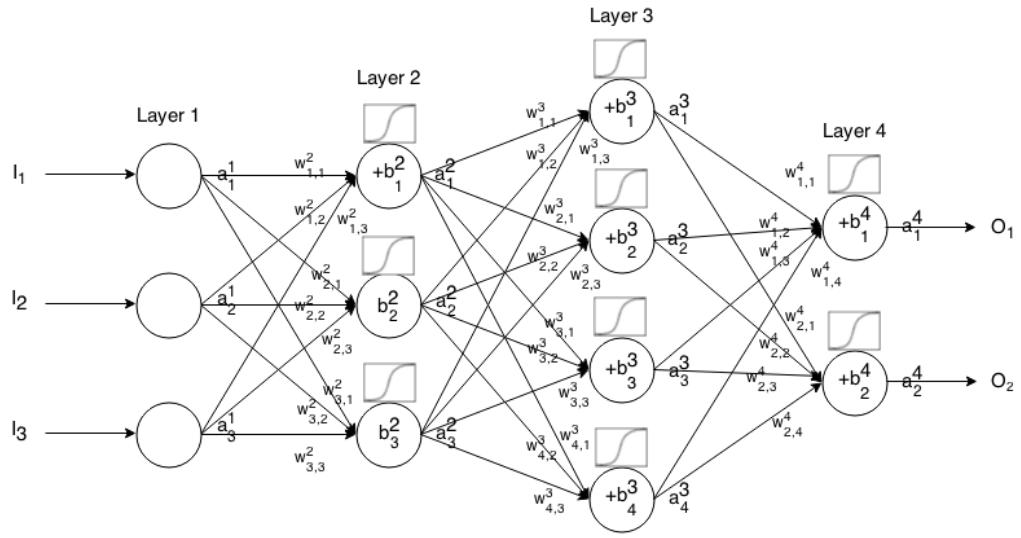
Zadnji razdelek 5 opisuje in primerja navadno inicializacijo in inicializacijo s pomočjo matrične faktorizacije. Opisane so tako dobre strani inicializacije, kot tudi slabosti. Inicializacija je testirana na dveh podatkovnih množicah. Pri prvi množici je cilj čim boljša klasifikacija ročno napisanih številk, pri drugi pa napovedovanje točnih ocen, s katerimi bi uporabniki ocenili določeno šalo.

Poglavje 2

Nevronske mreže

Nevronske mreže spadajo pod sodobne metode za obdelavo in napovedovanje podatkov, ki navdih črpajo iz delovanja človeških in živalskih možganov. V zadnjih letih so doživele ponoven hiter razvoj, saj so bili odkriti novi algoritmi, kot so RBM [9], Dropout [17], ReLu [7] in Batch normalization [11], ki precej izboljšajo kakovost napovedovanja in točnost, ter izboljšana strojna oprema, ki zagotavlja hitrejše učenje.

Delovanje nevronskih mrež je okvirno sestavljeno iz večih korakov. Najprej iz vhodnih podatkov in uteži izračunamo izhodne aktivacije na zadnjem nivoju. Ta postopek se imenuje razširjanje naprej (angl. feed-forward) in je opisan v razdelku 2.1. Glede na to, ali izvajamo klasifikacijo ali regresijo, s pomočjo ustreznih funkcij, ki so na voljo, izračunamo napako nevronske mreže pri napovedovanju. Dve izmed najbolj popularnih funkcij sta opisani v razdelkih 2.2.1 in 2.2.2. Vrednost napake na izhodnih nevronih nato vzratno širimo nazaj do vhodnega nivoja z algoritmom, imenovanim vzratno razširjanje napake (angl. backpropagation of error), opisanim v razdelku 2.2. S tem dobimo gradiente, s pomočjo katerih optimiziramo uteži za boljši rezultat v prihodni iteraciji.



Slika 2.1: Prikaz preproste štiri nivojske nevronske mreže [1].

Slika 2.1 prikazuje preprosto nevronsko mrežo z dvema skritima nivojema, tremi vhodi in dvema izhodoma. V nadaljnjih razdelkih bodo vse notacije in simboli skladni s to sliko.

Simboli:

$\sigma()^i$ ... aktivacijska funkcija za nivo i

w^i_{jk} ... utež med nevronom k v nivoju $i-1$ do nevrona j v nivoju i

b^i_j ... pristranska vrednost nevrona j v nivoju i

a^i_j ... izhod nevrona j v nivoju i

a^L_j ... izhod nevrona j na izhodu nevronske mreže

$\mathbf{W}^i$ ... matrika uteži, ki preslika vrednosti iz nivoja $i-1$ v nivo i

$\mathbf{b}^i$ ... vektor pristranskih vrednosti, ki preslika vrednosti iz nivoja $i-1$ v nivo i

2.1 Napovedovanje z razširjanjem naprej

Razširjanje naprej je postopek pri učenju nevronske mreže, kjer vhodno matriko podatkov s pomočjo uteži in umetnih nevronov transformiramo v drugo

matriko na izhodu, kjer je cilj, da se matrika na izhodu čim bolj ujema z podatki, ki jih napovedujemo. Nevroni, ki so del vsake nevronske mreže, so preproste funkcije, ki na vhod prejmejo uteži in vrednosti in iz njih izračunajo utežene izhode, ki jih nato normalizirajo še z eno od aktivacijskih funkcij, ki so opisane v razdelku 2.3. Skoraj vedno se izhod nevrona izračuna s formulo:

$$z_j^i = \sum_k (w_{jk}^i a_k^{i-1}) + b_j^i \tag{2.1}$$

ter nato še normalizira z:

$$a_j^i = \sigma^i(z_j^i) \tag{2.2}$$

Obstajajo tudi druge vrste nevronov, kjer utežene vrednosti med seboj množimo, vendar ti nevroni nikoli niso zaživeli v praksi in se običajno izkažejo kot slabša alternativa.

Nevrone lahko med seboj povežemo na več načinov. Eden izmed njih je tudi ta, da nevrone povežemo v cikle, vendar se tej tehniki v diplomski nalogi izognemo. Najbolj običajna arhitektura povezovanja je združevanje nevronov v polno povezane zaporedne nivoje, kjer je vsak nevron povezan z vsemi nevroni na prejšnjem nivoju in nivoju, kateremu sledi, ter z nobenim drugim nevronom iz katerega koli drugega nivoja ali z nevroni znotraj svojega nivoja.

Algoritem razširjanja naprej je precej kompleksen v primeru, da nevronsko mrežo programiramo kot skupek posameznih nevronov in vsak nevron obravnavamo kot posamezno funkcijo, ločeno od drugih nevronov. Če pristranske vrednosti na vsakem nivoju pametno združimo z matriko uteži in podatkom na vsakem nivoju dodamo dodaten stolpec enic, potem lahko izrabimo matrični zapis in izračun aktivacij od začetnega nivoja pa vse do končnega izhodnega nivoja preprosteje zapišemo kot zmnožek matrik:

$$\sigma^i(\mathbf{W}^i\sigma^{i-1}(\mathbf{W}^{i-1}\dots\sigma^1(\mathbf{W}^1\mathbf{X})))$$

Pri tem ni potrebno, da so aktivacijske funkcije na posameznih nivojih enake. Pogosto se na skritih nivojih od 1 do $i-1$ uporabijo sigmoidna, tangentna ali ReLu-funkcija, zadnji i nivo pa uporablja funkcijo glede na

tip problema. Če je problem klasifikacijski, potem je na zadnjem nivoju največkrat sigmoidna funkcija ali fukcija imenovana softmax, za regresijske probleme pa se uporabi linearna funkcija.

2.2 Vzratno razširjanje napake

Ker so nevronske mreže z naključno nastavljenimi utežmi brez učenja skoraj neuporabne, se za učenje uporablja algoritem imenovan vzratno razširjanje napake. Vzratno razširjanje napake je postopek, kjer iterativno posodabljamo vrednosti uteži v smeri zmanjšanje napake in s tem poskušamo izboljšati napovedno točnost nevronske mreže.

Pri vzratnem razširjanju napake je potrebno najprej definirati funkcijo napake, ki jo bomo minimizirali. V nadaljevanju je funkcija napake za primer x iz podatkovne množice definirana kot $C_x(a^L, y\prime)$, kjer vektorja a^L in $y\prime$ predstavljata trenutni izhod na nevronski mreži ter želen izhod nevronske mreže. Dve najbolj pogosti funkciji napake sta definirani v razdelkih 2.2.1 in 2.2.2. Vrednost napake na učni množici, ki je večja od 1 in jo imenujemo paketno učenje (angl. batch learning), se izračuna z:

$$C = \frac{1}{n} \sum_x C_x(a^L, y\prime) \tag{2.3}$$

kjer je n število učnih primerov. To napako nato uporabimo pri računanju gradienta, pri katerem si pomagamo z vmesnimi koraki. Najprej izračunamo spreminjanje napake $C_x(a^L, y\prime)$ na izhodu nevronske mreže glede na spreminjanje z_j^L vrednosti nevronov. Spreminjanje napake na zadnjem izhodnem nivoju zato definiramo kot:

$$\delta^L = \frac{\partial C}{\partial z^L} = \frac{\partial C}{\partial a^L} \odot \sigma^{L\prime}(z^L) \tag{2.4}$$

in za vse druge skrite nivoje v nevronski mreži:

$$\delta^i = \frac{\partial C}{\partial z^i} = ((\mathbf{W}^{i+1})^T \delta^{i+1}) \odot \sigma^{i\prime}(z^i) \tag{2.5}$$

Simboli:

$\frac{\partial C}{\partial a^L}$ ... predstavlja vektor odvodov funkcije napake

$\sigma^{i'}()$ ... odvod aktivacijske funkcije na posameznih nivojih

$\odot$ ... Hadamard produkt med dvema matrikama. $(x \odot y)_{i,j} = x_{i,j} * y_{i,j}$

Da dobimo gradient največjega padca funkcije, ki ga uporabimo za minimizacijo uteži, je potrebno napako nato distribuirati med uteži in pristranske vrednosti po enačbi:

$$\frac{\partial C}{\partial w^i_{jk}} = a^{i-1}_k \delta^i_j \tag{2.6}$$

$$\frac{\partial C}{\partial b^i_j} = \delta^i_j \tag{2.7}$$

S tem dobimo matriko gradientov za posamezno matriko uteži in pristranskih vrednosti:

$$\Delta \mathbf{W}^i = \begin{bmatrix} \frac{\partial C}{\partial w^i_{1,1}} & \cdots & \frac{\partial C}{\partial w^i_{1,k}} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial w^i_{j,1}} & \cdots & \frac{\partial C}{\partial w^i_{j,k}} \end{bmatrix} \tag{2.8}$$

$$\Delta \mathbf{b}^i = \begin{bmatrix} \frac{\partial C}{\partial b^i_1} \\ \vdots \\ \frac{\partial C}{\partial b^i_j} \end{bmatrix} \tag{2.9}$$

Za gradientni spust lahko uporabimo več različnih metod. Ena izmed najbolj preprostih je posodabljanje uteži in pristranskih vrednosti po formuli:

$$\mathbf{W}^i \leftarrow \mathbf{W}^i - \alpha * \Delta \mathbf{W}^i \tag{2.10}$$

$$\mathbf{b}^i \leftarrow \mathbf{b}^i - \alpha * \Delta \mathbf{b}^i \tag{2.11}$$

kjer konstanta α predstavlja hitrost učenja, ki jo nastavimo pred začetkom učenja.

Poznamo še več drugih metod za optimizacijo uteži nevronske mreže. Nekatere funkcije med optimizacijo nabirajo moment, kar pospeši premik

uteži v pravo smer in zmanjša oscilacijo, če ima več zaporednih premikov isti predznak. Ena izmed izboljšav navadnega gradientnega spusta z momentom je metoda Adam (angl. adaptive moment estimation) [18], kjer ima med procesom optimizacije vsaka utež svojo hitrost učenja in svoj moment. Ta metoda je uporabljena pri vseh rezultatih v diplomski nalogi, saj je trenutno ena izmed hitrejših metod.

2.2.1 Povprečna kvadratna napaka

Za regresijske probleme običajno uporabimo funkcijo, imenovano napaka kvadratov razlik (angl. mean square error). Funkcija je za regresijske probleme največkrat oblike:

$$C_x(a^L, y') = 0.5 \sum_j (a_j^L - y'_j)^2 \tag{2.12}$$

kjer a_j^L predstavlja j-ti izhod na izhodnem nivoju nevronske mreže iz slike 2.1. Če želimo nevronsko mrežo bolj kaznovati za večje napake, se občasno uporabi tudi potenciranje razlike na višje sode potence. Mera kvadratov napak ima preprost odvod definiran z:

$$\frac{\partial}{\partial a_j^L} C_x(a^L, y') = (a_j^L - y'_j) \tag{2.13}$$

pri tem pa ni nobenih omejitev na velikost števil, kot je to pri napaki prečne entropije, saj ni nujno, da vrednosti na izhodu predstavljajo verjetnosti.

2.2.2 Prečna entropija

Pri klasifikaciji se največkrat uporablja mera napake, imenovana prečna entropija (angl. cross-entropy), ki ima to prednost, da pospeši hitrost učenja v primerjavi z povprečno kvadratno napako. Funkcija je definirana z enačbo:

$$C_x(a^L, y') = -\sum_j [y'_j \ln a_j^L + (1 - y'_j) \ln (1 - a_j^L)] \tag{2.14}$$

njen odvod pa z:

$$\frac{\partial}{\partial a_j^L} C_x(a^L, y') = \frac{(a_j^L - y'_j)}{(1 - a_j^L)(a_j^L)} \tag{2.15}$$

Pri meri prečne entropije je potrebno, da velja $\forall_j(0 \leq a_j^L \leq 1)$, saj vrednosti na izhodih predstavljajo verjetnost pripadnosti določenemu razredu.

2.3 Aktivacijske funkcije

Aktivacijske funkcije so matematične funkcije, ki se aplicirajo na izhod vsakega nevrona. Poznamo več vrst aktivacijskih funkcij in so pomemben del nevronskih mrež, saj se nevronske mreže brez njih ne morejo naučiti nelinearne preslikave iz vhodnih spremenljivk v izhodne spremenljivke. Brez aktivacijske funkcije je vsaka globoka mreža enaka linearni transformaciji, kar pa ni zaželjeno. Neučinkovitost mreže brez aktivacijskih funkcij lahko pokažemo z naslednjo enačbo, kjer matriki $\mathbf{W}^i$ in $\mathbf{X}$ predstavljata uteži na i-tem nivoju in matriko vhodnih podatkov:

$$(\mathbf{W}^i(\mathbf{W}^{i-1}...(\mathbf{W}^1\mathbf{X}))) = (\mathbf{W}^i\mathbf{W}^{i-1}...\mathbf{W}^1)\mathbf{X} = \mathbf{W}\mathbf{X} \tag{2.16}$$

2.3.1 Sigmoidna funkcija

Sigmoidna funkcija je ena izmed prvih uspešno uporabljenih aktivacijskih funkciji, saj rešuje problem binarne aktivacije, katere pomankljivost je ta, da ni zvezna in odvedljiva, ter linearne funkcije, kjer ni možno zgraditi nevronske mreže, ki preslika vhod v nelinearen izhod. Sigmoidna funkcija je definirana z enačbo:

$$sigm(x) = \frac{1}{1 - e^{-x}} \tag{2.17}$$

njen odvod pa z:

$$\frac{\partial}{\partial x} sigm(x) = sigm(x) * (1 - sigm(x)) \tag{2.18}$$

Iz enačbe, ki definira sigmoidno funkcijo, je razvidno, da je definicijsko območje $(-\infty, \infty)$, in zaloga vrednosti $(0, 1)$, kar daje nevronom lepo lastnost, saj je njihov izhod vedno normaliziran med 0 in 1.

Sigmoidna aktivacijska funkcija ima tudi dve večji težavi. Prva izmed njih je majhen gradient, ko je $x \ll 0$ ali $x \gg 0$, saj velja:

$$\lim_{x \to \pm\infty} \frac{\partial}{\partial x} sigm(x) = 0 \tag{2.19}$$

kar pomeni, da je pri inicializaciji uteži potrebna previdnost, saj je lahko vhod v sigmoidno funkcijo tako velik, da je odvod skoraj 0, kar zelo upočasni učenje pri vzvratnem razširjanju napake.

Druga težava sigmoidne funkcije pa je, da je izhod funkcije lahko le pozitivno realno število, zato so posodobitve vseh uteži za določen nevron lahko le vse pozitivne oziroma vse negative, kar pripelje do osilacije uteži.

2.3.2 Tangentna funkcija

Tangentna funkcija je aktivacijska fukcija, ki je zelo podobna sigmoidni funkciji, zato ima tudi podobne lastnosti in težave. Definirana je kot:

$$tanh(x) = \frac{2}{1 + e^{-2x}} + 1 \tag{2.20}$$

oziroma skalirana sigmoidna funkcija:

$$tanh(x) = 2sigm(2x) - 1 \tag{2.21}$$

njen odvod pa z:

$$\frac{\partial}{\partial x} tanh(x) = 1 - tanh(x)^2 \tag{2.22}$$

Tangentna funkcija se srečuje s podobno težavo kot sigmoidna funkcija, saj ima pri vrednostih $x \ll 0$ ali $x \gg 0$ težave z učenjem, ker je odvod skoraj enak nič, kar zelo upočasni vzratno propagiranje napake.

Prednost funkcije tanh je ta, da vhod iz intervala $(-\infty, \infty)$, pretvori v vrednosti na intervalu $(-1, 1)$ in se s tem izogne problemu sigmoidne funkcije, kjer so uteži ob posodobitvi vse pozitivne ali vse negativne.

2.3.3 Funkcija ReLu

ReLu je ena izmed novejših aktivacijskih funkcij, ki se izogne problemu izginjajočega gradienta, ki jih imata funkciji sigm in tanh vendar ime nekatere

druge probleme, ki jih sigmoidna in tanh funkciji nimata. Definirana je kot:

$$relu(x) = max(0, x) \tag{2.23}$$

z odvodom:

$$\frac{\partial}{\partial x} relu(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{2.24}$$

kjer odvod v točki 0 ni definiran, vendar to v praksi ne predstavlja težav, saj lahko v primeru vhoda 0, vrnemo 0 ali 1. Iz enačbe je razvidno, da je definicijsko območje funkcije na intervalu $(-\infty, \infty)$, zaloga vrednosti pa na intervalu $[0, \infty)$.

Največja prednost aktivacijske funkcije relu je ta, da je v primeru, ko je $x \gg 0$, odvod funkcije še vedno 1, kar precej pospeši učenje. Prednost je tudi redka predstavitev znanja na izhodih nevonih, saj se med učenjem nekateri nevroni povsem izključijo oziroma imajo na izhodu vrednost 0 in pri napovedovanju ne vplivajo na rezultat, kar se pri drugih aktivacijah ne zgodi pogosto.

Slabost aktivacijske funkcije relu so tako imenovani umirajoči nevroni. Ti nastanejo med učenjem ali zaradi slabe inicializacije uteži. To so nevroni, ki se jim med učenjem vrednost zmanjša pod 0 in se s tem prenehajo učiti. Obstaja nekaj variant, ki poskušajo odpraviti to težavo. Ena izmed bolj znanih variant funkcije ReLu je leaky ReLu. Ta se med učenjem spreminja, tudi, če je izhod manjši od 0.

2.4 Regularizacija

Pri učenju z nevronskimi mrežami največkrat pride do prevelikega prileganja učni množici, sploh če imajo nevronske mreže veliko nevronov, zato je pomembno, da se prevelikemu prileganju poskušamo izogniti. Ena izmed najbolj enostavnih tehnik za izogib prevelikemu prileganju je, da za učenje

pridobimo večjo količino podatkov. To lahko predstavlja težavo, če je pridobivanje večje količine podatkov zahteven in drag postopek. Druga možnost je, da podatke umetno ustvarimo iz že obstoječih podatkov. Primer takšnega ustvarjanja podatkov je na primer rotacija, razteg in zrcaljenje slik, vendar ta tehnika na vseh podatkih ni mogoča, zato uporabimo regularizacijo.

Regularizacija je postopek, ki skuša zmanjšati kompleksnost naučene funcije in pri tem čim manj zmanjšati točnost na učnih podatkih. Osnova regularizacije je princip najkrajšega opisa (MDL), ki pa je neizračunljiv in se zato v praksi uporabljajo različne hevristike. Večinoma so usmrejene v to, da funcija, ki jo želimo minimizirati, ne vključuje samo napovedne napake ampak tudi kompleksnost funkcije. Na ta način hkrati minimiziramo napako in kompleksnost funkcije (pri nevronski mreži je to število in velikost uteži).

Tipične regularizacije, ki jih uporabimo pri učenju so L1, L2 in Dropout. Regularizaciji L1 in L2 sta si med seboj zelo podobni, saj kaznujeta prevelike uteži. Pri Dropout regularizaciji med razširjanjem naprej določenim nevronom nastavimo izhod na nič. Občasno se uporablja še regularizacija z zgodnjim ustavljanjem, kjer nevronsko mrežo učimo, dokler se napaka ne validacijski množici ne začne povečevati.

2.4.1 L1

Regularizacija L1 je ena izmed najbolj preprostih regularizacij, kjer ceni napake prištejemo še absolutne vrednosti vseh uteži. Celotna napaka je v tem primeru definirana kot:

$$C_r = C + \frac{\lambda}{2n} \sum_w |w| \tag{2.25}$$

njen odvod pa kot:

$$\frac{\partial C_r}{\partial w} = \frac{\partial C}{\partial w} + \frac{\lambda}{n} \operatorname{sgn}(w) \tag{2.26}$$

Prednost regularizacije L1 je ta, da med učenjem ustvari redko matriko uteži (angl. sparse matrix), kar pomeni, da uteži spodbudi k temu, da se med učenjem nastavijo na nič oziroma da deluje kot algoritem za izbor atributov

(angl. feature selection). Regularizacija L1 za razliko od regularizacije L2 ni definirana tako, da bi imela le eno analitično rešitev.

2.4.2 L2

Regularizacija L2 je precej podobna regularizaciji L1, le da tu kaznujemo prvotno ceno napake še s seštevkom kvadratov vseh uteži. Napaka je zato definirana kot:

$$C_r = C + \frac{\lambda}{2n} \sum_w w^2 \tag{2.27}$$

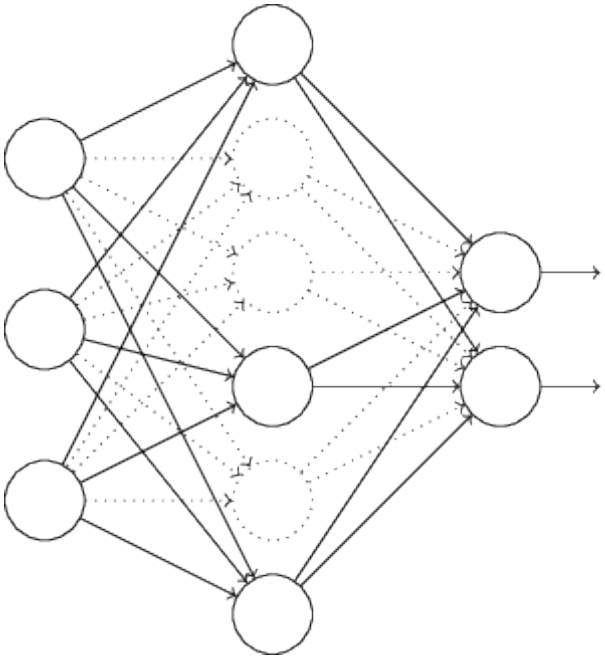
odvod napake pa z:

$$\frac{\partial C_r}{\partial w} = \frac{\partial C}{\partial w} + \frac{\lambda}{n} w \tag{2.28}$$

Največja prednost te regularizacije je, da običajno deluje bolje kot regularizacija L1, vendar uteži skoraj nikoli ne nastavi na nič in se s tem nauči redke predstavitve podatkov. Regularizacija L2 vedno preferira razpršeno (angl. diffused) matriko uteži, kjer so vse uteži majhne, nobena pa ni enaka nič, in precej kaznuje matriko uteži, kjer so posamezne vrednosti precej velike, vendar je večina uteži nastavljenih na nič.

2.4.3 Dropout

Dropout [10] je ena novejših tehnik regularizacije, kjer med učenjem izključimo večje število nevronov in s tem prisilimo nevrone, da se naučijo redundantno znanje, saj se ne morejo vedno zanašati na izhod nevronov v prejšnjem nivoju. Število izključenih nevronov običajno nastavimo do 25% za vhodni nivo in do 50% za skrite nivoje.



Slika 2.2: Prikaz delovanja tehnike Dropout na nevronski mreži z enim skritim nivojem in 50% verjetnostjo za izključitev nevronov [2].

Slika 2.2 prikazuje algoritem, ki najprej izvede postopek razširjanja naprej in na vsakem nivoju določenim nevronom nastavi izhod na nič z verjetnostjo p in s tem prikrajša naslednji nivo za določene podatke, ki bi jih nevroni dobili v popolno povezani mreži. Ta postopek se nadaljuje do zadnjega nivoja, nakar se isto kot v normalnih mrežah izvede vzratno razširjanje napake nazaj do vhodnega nivoja.

Pri kasnejšem napovedovanju podatkov, kjer uporabimo vse nevrone, je potrebno aktivacije na posameznih nivojih pomnožiti še z $\frac{1}{p}$, saj se je mreža prvotno naučila delovanja le z delom nevronov, pri predikciji pa so upore-bljani izhodi vseh nevronov prejšnjega nivoja. Če to skalacijo izvedemo že med učenjem, potem to verzijo regularizacije imenujemo Inverted Dropout.

Poglavje 3

Matrična faktorizacija

Matrična faktorizacija je postopek, kjer poskušamo matriko razdeliti na dve ali več manjših matrik, ki ob ponovnem zmnožku kar se da natančno ocenijo vrednosti prvotne matrike. Ti postopki so v zadnjem času postali precej popularni na večih področjih strojnega učenja, saj so na nekaterih področjih precej izboljšali uspešnosti modelov [13] [16]. Z matrično faktorizacijo lahko iz podatkov, organiziranih v matrično tabelo, dobimo nove tabele, v katerih so predstavljeni zgoščeni podatki osnovne matrike. Zgoščeni podatki so lahko uporabni na več načinov:

- Izračun mankajočih podatkov
  Skoraj v vseh problemih, s katerimi se srečuje strojno učenje, večkrat naletimo na nepopolne ali mankajoče podatke. S pomočjo matrične faktorizacije lahko za nekatere podatke ponovno ocenimo, kakšno vrednost bi morali imeti. Primer izračuna podatkov s pomočjo matrične faktorizacije je napovedovanje ocen filmov, ki si jih uporabniki še niso ogledali. Če matriki, ki nastaneta pri razcepu z nenegativno matrično faktorizacijo ponovno zmnožimo, potem nova matrika predstavlja ocene, ki naj bi jih uporabniki dali posameznim filmom.

- Zmanjšanje dimenzij podatkov
  Matrično faktorizacijo lahko uporabljamo kot nanadzorovano tehniko

za zmanjšanje dimenzije podatkov. Prvotno matriko s pomočjo matrične faktorizacije razdelimo v nove matrike, ki imajo manjše dimenzije od originalne matrike. Pri tem nove matrike predstavljajo stisnjene podatke oziroma povzetek podatkov v prvotni matriki. Te povzetke podatkov lahko nato uporabimo za nadaljnjo obdelavo z različnimi algoritmi strojnega učenja.

- Gručenje podatkov
  Matrična faktorizacija lahko služi tudi kot algoritem za iskanje gruč v podatkih. Pred začetkom faktorizacije izberemo dimenzijo oziroma velikost matrik, ki bodo nastale pri faktorizaciji. Ta velikost kasneje predstavlja število gruč, ki jih iščemo v podatkih. Ena izmed matrik po končani faktorizaciji nato predstavlja centre posameznih gruč, medtem ko druga matrika prikazuje pripadnost posameznega primera določeni grupi.

3.1 Opis delovanja

Ob vhodni matriki $\mathbf{X} \in \mathbb{R}^{n \times m}$ iščemo dve matriki $\mathbf{W}$ in $\mathbf{H}$, kjer velja $\mathbf{X} \approx \mathbf{WH} = \mathbf{X}'$, pri čemer $\mathbf{W} \in \mathbb{R}^{n \times k}$ in $\mathbf{H} \in \mathbb{R}^{k \times m}$. Pri faktorizaciji je velikost novih matrik označenih z velikostjo k ter velja $k \ll min(n, m)$. Glede na tip matrične faktorizacije lahko matriko $\mathbf{X}$ razdelimo tudi na več kot dve manjši matriki. Primer takšne faktorizacije je nenegativna matrična trifaktorizacija.

Pri iskanju matrik $\mathbf{W}$ in $\mathbf{H}$ je potrebno definirati še funkcijo kakovosti rešitve, ki jo optimiziramo. Največkrat je to Frobeniusova razdalja oziroma razdalja med razlikami kvadratov napak. Razdalja med matriko $\mathbf{X}$ in matriko $\mathbf{X}'$ je definirana kot:

$$\|\mathbf{X} - \mathbf{X}'\|_F^2 = \|\mathbf{X} - \mathbf{WH}\|_F^2 = \sum_{nm} (\mathbf{X}_{nm} - \mathbf{X}'_{nm})^2 \tag{3.1}$$

Ker iščemo najboljšo aproksimacijo matrike $\mathbf{X}$ s pomočjo matrik $\mathbf{W}$ in

H, lahko problem optimizacije zastavimo kot:

$$min\|\mathbf{X} - \mathbf{WH}\|_F^2 \tag{3.2}$$

kar pomeni, da minimiziramo razliko med matrikama oziroma zmanjšujemo napako zmnožka matrik **W** in **H**.

3.2 Tipi matrične faktorizacije

Obstaja več tipov matrične faktorizacije. Faktorizacije se razlikujejo po funkciji, ki jo optimizirajo, številom novih matrik, ki jih ustvarijo, in različnih omejitvah, ki se lahko pojavijo znotraj posameznih matrik. Za enake tipe matrične faktorizacije obstaja tudi več različnih aproksimacijskih algoritmov, ki se razlukujejo po kompleksnosti in hitrosti konvergence. V naslednjih razdelkih so predstavljeni tipi matrične faktorizacije, ki so bili uporabljeni v diplomski nalogi.

3.2.1 Nenegativna matrična faktorizacija - NMF

Nenegativna matrična faktorizacija (angl. non-negative matrix factorization) je matrična faktorizacija, ki ima dodatno omejitev na originalni matriki **X** in posledično še na obeh matrikah **W** in **H**. Vse tri matrike lahko vsebujejo le pozitivne vrednosti oziroma velja:

$$\mathbf{X}_+ \approx \mathbf{W}_+\mathbf{H}_+ \tag{3.3}$$

Matrika **W** v tem primeru predstavlja linearno kombinacijo koeficientov zapisanih v matriki **H**. Za to metodo matrične faktorizacije pravimo, da je aditivna, saj so vsi elementi v matrikah pozitivni in jih med seboj nikoli ne odštevamo. Pogosto se uporablja pri podatkih, ki so že pri nastanku pozitivni, saj jih pred faktorizacijo ni potrebno skalirati. Taki podatki so na primer RGB-vrednosti pikslov na sliki int npr. ocene, ki so jih uporabniki dali določenim filmom.

Eden izmed najbolj preprostih algoritmov je iterativno posodabljanje matrik **W** in **H** po enačbi:

$$H_{ij} \leftarrow H_{ij} \frac{(W^T X)_{ij}}{(W^T W H)_{ij}} \tag{3.4}$$

$$W_{ij} \leftarrow W_{ij} \frac{(X H^T)_{ij}}{(W H H^T)_{ij}} \tag{3.5}$$

Ta metoda se imenuje multiplikativno posodabljanje uteži [15] in je ena izmed prvih metod za razcep matrik. Obstaja pa še več različnih novejših metod, ki različno hitro konvergirajo k lokalnem minimumu.

3.2.2 Semi-nenegativna matrična faktorizacija - SNMF

Semi-nenegativna matrična faktorizacija (angl. semi-nonnegative matrix factorization) je tip matrične faktorizacije, kjer dopuščamo, da ima matrika **W** tako pozitivne kot tudi negativne vrednosti in je zelo podobna nenegativni matrični faktorizaciji s to razliko, da velja:

$$\mathbf{X}_{\pm} \approx \mathbf{W}_{\pm} \mathbf{H}_{+} \tag{3.6}$$

Ker dopuščamo negativne vrednosti v matriki **W**, lahko matriko **W** obravnavamo kot centre gruč, algoritem pa podobno kot druge algoritme za gručenje. V nekaterih pogledih je algoritem SNMF podoben algoritmu k-means, kjer iščemo k gruč, ki so na danih podatkih najbolj smiselne. Ker se v realnosti večkrat pojavijo podatki z negativnimi vrednostmi, je prednost SNMF pred NMF ta, da lahko z algoritmom SNMF razcepimo tudi matrike, ki vsebujejo negativne vrednosti in se s tem izognemo skaliranju podatkov, ki bi jih zahteval algoritem nenegativne matrične faktorizacije.

3.2.3 Singularni razcep - SVD

Singularni razcep (angl. singular value decomposition) je postopek, pri katerem prvotno matriko $\mathbf{X} \in \mathbb{R}^{n \times m}$ razdelimo v tri nove matrike in s pomočjo

teh treh matrik zmanjšamo dimenzije podatkov. Pri singularnem razcepu zato iščemo matrike $\mathbf{U} \in \mathbb{R}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ in $\mathbf{V}^\top \in \mathbb{R}^{m \times m}$, kjer velja:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top \tag{3.7}$$

Matrika $\mathbf{\Sigma}$ je definirana kot matrika, ki ima na diagonali korene lastnih vrednosti (angl. eigenvalues), dobljene iz razcepa matrike $\mathbf{X}^\top\mathbf{X}$, v matriki $\mathbf{V}$ pa se nahajajo normalizirani lastni vektorji (angl. eigenvectors), dobljeni z istim razcepom in zloženi vsak v svoj stolpec. Zadnjo matriko $\mathbf{U}$ dobimo iz enačbe $\mathbf{X}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}$, kjer so vrednosti $\mathbf{X}$, $\mathbf{V}$ in $\mathbf{\Sigma}$ že znane.

Iz delovanja SVD lahko hitro ugotovimo, da je v matrikah precej vrstic, ki na izhod nimajo vpliva ali pa je ta vpliv zelo majhen. Kako določen stolpec v matrikah $\mathbf{U}$ in $\mathbf{V}$ vpliva na matriko $\mathbf{X}$, lahko vidimo iz diagonalnih elementov v matriki $\mathbf{\Sigma}$. Diagonalni elementi predstavljajo uteži vektorjev pri rekonstrukciji prvotne matrike, vendar, ker so nekatere izmed teh uteži precej majhne, jih lahko nastavimo na 0. Če nekatere vrednosti na diagonali nastavimo na 0, je to isto, kot če bi pri množenju matrik izpustili določene stolpce.

Ker so vrednosti na diagonali $\mathbf{\Sigma}$ urejene po velikosti, lahko izbrišemo zadnjih nekaj vrstic in stolpcev matrike $\mathbf{\Sigma}$, ki vsebujejo vrednosti skoraj enako nič. Nova matrika ima drugačne dimenzije, zato je potreben izbris še isto število stolpcev iz matrike $\mathbf{U}$ in vrstic iz matrike $\mathbf{V}$. S tem postopkom dobimo nove matrike $\mathbf{U} \in \mathbb{R}^{n \times k}$, $\mathbf{\Sigma} \in \mathbb{R}^{k \times k}$ in $\mathbf{V}^\top \in \mathbb{R}^{k \times m}$, ki uporabljajo manj podatkov za rekonstrukcijo originalne matrike, v njih pa je zgoščena predstavitev podatkov.

Običajno je nova matrika, ki predstavlja približek originalne matrike, sestavljena iz dveh matrik, ki jih dobimo pri matrični faktorizaciji. V primeru manjšanja dimenzij je matrika $\mathbf{H}$ enakovredna matriki $\mathbf{V}^\top$, matriko $\mathbf{W}$ pa zapišemo kot zmnožek $\mathbf{U}$ in $\mathbf{\Sigma}$.

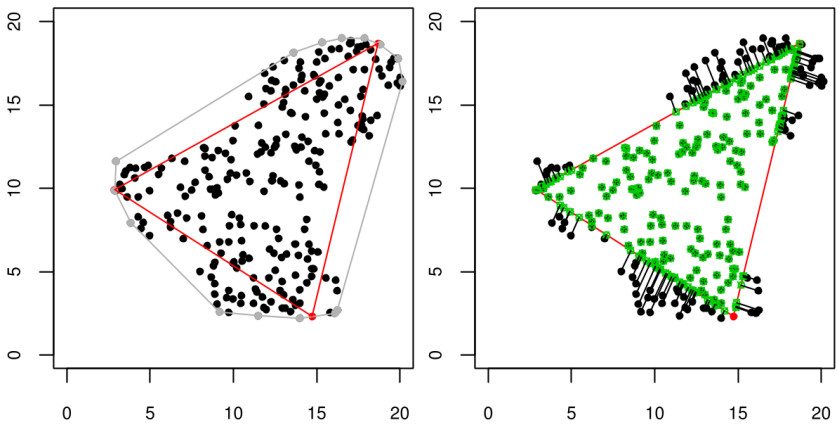
$$\mathbf{X}_\pm \approx \mathbf{W}_\pm\mathbf{H}_\pm \tag{3.8}$$

$$\mathbf{W} = \mathbf{U}\mathbf{\Sigma} \tag{3.9}$$

$$\mathbf{H} = \mathbf{V}^\top \tag{3.10}$$

3.2.4 Analiza arhetipov - AA

Analiza arhetipov (angl. archetypal analysis) [4] je postopek, ki je podoben matrični faktorizaciji, kjer iterativno iščemo manjše število arhetipov, ki predstavljajo konveksno kombinacijo originalnih podatkov. Analizo arhetipov lahko obravnavamo tudi kot razcep podatkov na 2 matriki, kjer matrika **W** predstavlja verjetnost pripadnosti posameznega primera določenemu arhetipu, matrika **H** pa atribute posameznega arhetipa. Najdeni arhetipi običajno tvorijo kar se da dobro konveksno ovojnico celotne množice podatkov, ostale podatke, ki se nahajajo izven te konveksne ovojnice, pa preslikamo na najbližjo točko znotraj ovojnice.



Slika 3.1: Primer algoritma analize arhetipov, kjer iščemo konveksno ovojnico iz treh arhetipov. Vse točke, ki so zunaj konveksne ovojnice nato preslikamo v ovojnico [5].

V kontekstu diplomske naloge je analiza arhetipov uporabljena predvsem kot algoritem za mehko gručenje (angl. soft clustering) oziroma verjetnostno rangiranje (angl. probabilistic ranking) ter za zmanjševanje dimenzij multivariantnih podatkov, kjer matriko prvotnih podatkov **X** razdelimo v matriko arhetipov in matriko verjetnosti pripadnosti posameznemu arhetipov. Analiza arhetipov ima pred drugimi algoritmi to prednost, da matrika $\mathbf{W} \in \mathbb{R}^{n \times k}$ po vrsticah vsebuje verjetnosti, kar pomeni da je seštevek vsake vrstice enak

1 oziroma velja:

$$\sum_k W_{nk} = 1$$

Prednost matrike, ki vsebujejo verjetnosti, je ta, da pri nekaterih aktivacijskih funkcijah ne pride do prevelike nasičenosti posameznih nevronov.

3.2.5 Analiza glavnih komponent - PCA

Analiza glavnih komponent (angl. principal component analysis) je metoda, ki preslika točke iz enega prostora v drugega z namenom, da bi podatkom zmanjšali dimenzije atributov in ob tem ohranili čim več koristnih informacij. Predpostavimo, da imamo matriko $\mathbf{X} \in \mathbb{R}^{n \times m}$, kjer konstanti n in m predstavljata število primerov in število atributov, ki jih ima vsak primer. Cilj algoritma je preslikati matriko $\mathbf{X}$ v novo matriko $\mathbf{X}' \in \mathbb{R}^{n \times k}$, kjer velja $k < m$ tako, da nova matrika $\mathbf{X}'$ predstavlja linearne kombinacije originalnih primerov in da maksimiziramo varianco preslikanih točk.

Postopek za izračun matrike $\mathbf{X}'$ je sestavljen iz štirih korakov. Najprej vsakemu elementu v matriki $\mathbf{X}$ odštejemo povprečje vsakega stolpca, v katerem se nahaja element in jih s tem premaknemo na območje, kjer imajo povprečje nič (angl. zero centered).

Naslednji korak je izračun kovariančne matrike. Naj bo $\mathbf{C} \in \mathbb{R}^{m \times m}$ matrika, kjer velja:

$$\mathbf{C}_{ij} = cov(\mathbf{X}_i, \mathbf{X}_j) \tag{3.11}$$

V matriki $\mathbf{C}_{ij}$ so zapisane koveriance med vsemi pari atributov iz podatkovne množice $\mathbf{X}$. Kovariančna matrika je zato vedno kvadratna in vsebuje m^2 kovarianc.

Iz matrike kovariance izračunamo lastne vrednosti in vektorje s pomočjo dekompozicije. Dobljene lastne vrednosti in vektorje nato uredimo po velikosti, saj s tem dobimo nove osi, ki najbolj ohranjajo varianco v podatkih. Ker podatkom zmanjšujemo število atributov, lahko sestavimo novo matriko $\mathbf{Y} \in \mathbb{R}^{m \times k}$. Matrika $\mathbf{Y}$ je sestavljena iz k lastnih vektorjev, ki imajo največje lastne vrednosti in opravlja preslikavo iz originalnih podatkov v novo množico

podatkov z manjšimi dimenzijami. Novo matriko $\mathbf{X}'$ dobimo z enačbo:

$$\mathbf{X}' \leftarrow \mathbf{XY} \tag{3.12}$$

3.2.6 Faktorska analiza - FA

Faktorska analiza (angl. factor analysis) je metoda dekompozicije originalne matrike, ki vsebuje izmerjene oziroma vidne atribute, v manjšo matriko, v kateri se nahajajo neizmerjene skrite spremenljivke (angl. unobserved latent variables). Vidni atributi so pri procesu faktorizacije predstavljeni kot linearne kombinacije skritih spremenljivk z dodanim členom, ki predstavlja napako.

Analiza faktorjev je po načinu delovanja precej podobna matrični faktorizaciji, kjer podobno kot pri tej metodi, tudi tam iščemo skrite faktorje, ki se pojavljajo v podatkih.

Poglavje 4

Inicializacija uteži

Inicializacija uteži v nevronski mreži je eden izmed najbolj pomembni dejavnikov, ki vplivajo na hitrost in uspešnost učenja, sploh če imajo te mreže večje število skritih nivojev. To lahko vidimo že s preprosto inicializacijo, kjer vse uteži v nevronski mreži nastavimo na naključno konstanto. V primeru da so vse uteži za posamezen nivo enake, pride pri vzratnem širjenju napake do istih posodobitev vseh nevronov, kar posledično pomeni, da se vsi nevroni naučijo isto funkcijo.

V literaturi je bilo do sedaj predstavljenih več različnih načinov inicializacije, ki dosegajo različne uspehe. Najpogostejša inicializacija je bila do nedavnega naključna inicializacija, kjer vse uteži nastavimo na majhne naključne vrednosti. Ta inicializacija je pri globokih nevronskih mrežah povzročila prenasičenost nevronov, zato se trenutno največkrat uporablja inicializacija Xavier [6], ki poskuša ohranjati konstantno varianco na izhodih nevronov skozi vse nivoje.

4.1 Naključna inicializacija

Ena izmed prvih inicializacij je bila naključna inicializacija, ki se je izkazala za uspešno na plitvih nevronskih mrežah, vendar se je slabo obnesla na bolj globokih mrežah. Pri naključni inicializaciji uteži vzorčimo iz normalne dis-

tribucije $\beta\mathcal{N}(0,1)$, kjer β predstavlja konstanto, ki jo določimo pred začetkom učenja. Izbor konstante β je zelo pomemben, saj lahko ob premajhni konstanti vsi nevroni v globjih nivojih na izhodu napovejo vrednosti, katerih varianca je blizu 0 in s tem upočasnijo vzvratno širjenje napake. Če je konstanta β prevelika, se zgodi, da se varianca podatkov med prehajanji med nivoji precej poveča, kar pomeni, da se funkcije, kot so sigmoidna in tangentna, prenasičijo in na izhodu napovedujejo le vrednosti 0 ali 1 za sigmoidno funkcijo oziroma -1 in 1 za tangetno funkcijo, kar ponovno privede do izgube gradienta. Ena izmed rešitev za inicializacijo uteži, kjer se izognemo potrebi po točnem nastavljanju parametrov, sta metodi Batch Normalization [11] in Self-Normalizing Neural Networks [12], kjer po napovedi vsakega nivoja izhode skaliramo in s tem preprečimo nasičenost nevronov.

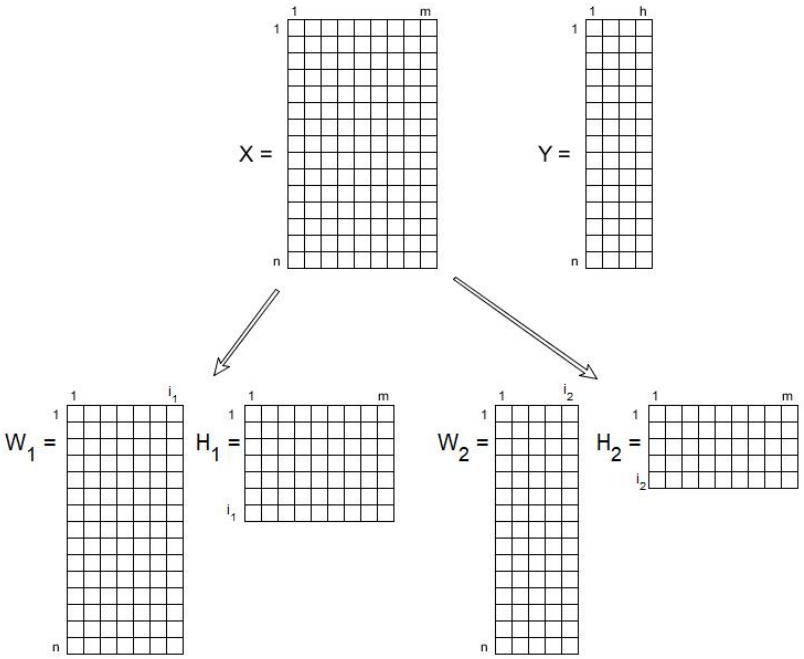
4.2 Inicializacija Xavier

Inicializacija Xavier je podobna naključni inicializaciji, le da ta rešuje problem izbora parametra β. Ta inicializacija poskuša na prehodih do globokih nivojev ohraniti varianco čim bolj konstanto in s tem preprečiti, da bi se aktivacijske funkcije prenasičile oziroma bi se varianca postopoma znižala do ničle.

Funkcija inicializacije uteži je odvisna tudi od aktivacijskih funkcij na izhodih vsakega nevrona, ker lahko nekatere funkcije porežejo del izhodov in s tem zmanjšajo varianco. Za sigmoidno, tangentno in funkcijo softmax se za inicializiranje uporablja vzorčenje iz distribucije $\frac{1}{\sqrt{n}}\mathcal{N}(0,1)$, kjer n predstavlja število vhodnih povezav v nevron, za katerega inicializiramo uteži. Pri funkciji ReLu je inicializacija nekoliko spremenjena, saj $max(0,x)$ na izhodih poreže polovico veljavnih vrednosti, zato je potrebno vzorčenje iz distribucije $\frac{1}{\sqrt{0.5n}}\mathcal{N}(0,1)$.

4.3 Predlagana inicializacija

V okviru diplomske naloge je predlagan nov način inicializacije, kjer s pomočjo tehnik za zmanjšanje dimenzij podatkov na vsakem skritem nivoju izračunamo določeno zgoščeno predstavitev vhodnih podatkov in z njo poskušamo pospešiti učenje. Ideja delno izhava iz nenadzorovanega učenja, kjer se na vsakem skritem nivoju poskušamo naučiti čim bolj abstraktno predstavitev podatkov, iz katerih lahko nato lažje pravilno napovemo vrednosti na izhodu.

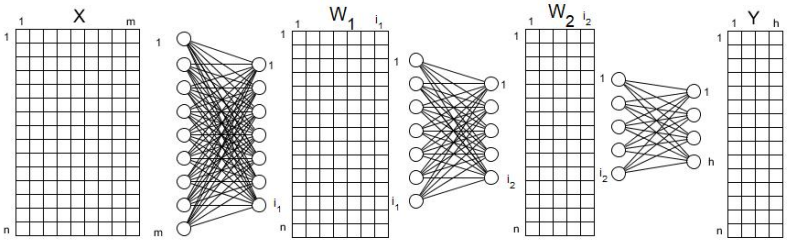


Slika 4.1: Prikaz razbitja originalne matrike na dva para manjših matrik, ki bosta predstavljala aktivacije na dveh skritih nivojih.

Predpostavimo, da imamo opravka s klasifikacijo, kjer so vhodni podatki v obliki matrike $\mathbf{X} \in \mathbb{R}^{n \times m}$, vrednosti, ki jih želimo napovedati pa so zakodirane v matriki $\mathbf{Y} \in \mathbb{R}^{n \times h}$ tako, da je za vsak primer enica v tistem stolpcu, kateremu razredu primer pripada, v vseh drugih stolpci pa so vrednosti nič (angl. one hot encoding). Konstante n, m in h predstavljajo število učnih

primerkov, dimenzijo učnih podatkov in število razredov pri klasifikaciji. Originalno matriko $\mathbf{X}$ najprej faktoriziramo v pare matrik $\{\mathbf{W}, \mathbf{H}\}$(slika 4.1), kjer vsak par predstavlja zgoščene podatke na posameznem skritem nivoju nevronske mreže.

Če to tehniko izvajamo na nevronski mreži iz slike 2.1 in se učimo na množici velikosti 10, imamo kot vhod matriko $\mathbf{X} \in \mathbb{R}^{10\times3}$, izhodne podatke pa oblike $\mathbf{Y} \in \mathbb{R}^{10\times2}$. Z matrično faktorizacijo ustvarimo dva para $\{\mathbf{W}_1 \in \mathbb{R}^{10\times3}, \mathbf{H}_1 \in \mathbb{R}^{3\times3}\}$ in $\{\mathbf{W}_2 \in \mathbb{R}^{10\times4}, \mathbf{H}_2 \in \mathbb{R}^{4\times3}\}$, ki predstavljata zgoščeno obliko izhoda na obeh skritih nivojih. Matriki $\mathbf{H}$ lahko zavržemo, saj nas zanima le zgoščena predstavitev znanja iz matrike $\mathbf{X}$.

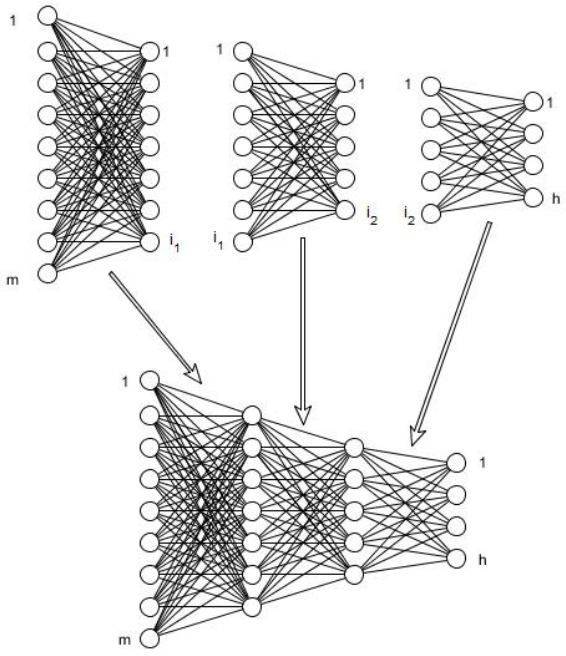


Slika 4.2: Učenje enonivojskih mrež, ki mapirajo vhodno matriko iz enega nivoja v izhodno matriko na naslednjem nivoju ciljne nevronske mreže.

V naslednjem koraku, prikazanem na sliki 4.2, zgradimo $k-1$ nevronskih mrež (k predstavlja število nivojev ciljne nevronske mreže), ki bodo delovale kot funkcije, ki preslikajo podatke, dobljene s pomočjo matrične faktorizacije za nivo i, v podatke, dobljene na nivoju $i+1$. Vse te nevronske mreže imajo le vhodni in izhodni nivo in nimajo nobenega skritega nivoja. Uteži teh $i-1$ mrež bodo kasneje uporabljene za gradnjo ciljne globoke nevronske mreže. Pred začetkom učenja teh enonivojskih mrež je potrebno določiti še aktivacijsko funkcijo na izhodnih nevronih ter morebitno skaliranje podatkov v matriki, saj nekatere aktivacijske funkcije ne morejo napovedati vrednosti na neomejenem intervalu. Aktivacijska funkcija je lahko sigmoidna ali softmax, če želimo na izhodih izračunavati verjetnosti ali linearna, oziroma ReLu, če

aktivacije na izhodih ne predstavljajo verjetnosti, omejene med nič in ena. Od izbora aktivacijske funkcije je odvisna tudi funkcija za izračun napake. Pri sigmoidni aktivaciji in aktivaciji softmax lahko za izračun napake uporabimo prečno entropijo, za linearno funkcijo in ReLu pa uporabljamo napako povprečne kvadratne razlike.

S to tehniko za primer nevronske mreže iz slike 2.1 zgradimo 3 nevronske mreže. Prva preslika podatke iz $\mathbf{X}$ v $\mathbf{W}_1$, druga iz $\mathbf{W}_1$ v $\mathbf{W}_2$, tretja pa iz $\mathbf{W}_2$ v $\mathbf{Y}$. Te tri mreže imajo na vhodih dimenzije 3, 3, 4, na izhodih pa 3, 4 in 2.



Slika 4.3: Združevanje delno naučenih enonivojskih mrež v ciljno nevronsko mrežo.

Vse nastale enonivojske mreže iz slike 4.2 nato združimo v ciljno globoko mrežo, kjer vsaka od enonivojskih mrež preslika svoj vhod v novo zgoščeno predstavitev znanja in to predstavitev poda nevronski mreži na naslednjem nivoju, kar prikazuje slika 4.3. Na zadnjem nivoju je potrebno, da se ne-

vronska mreža nauči preslikavo iz zgoščenih podatkov v podatke pripadnosti razredov oziroma, da zna na izhodu čim bolj natančno napovedati matriko $\mathbf{Y}$. Ko je ciljna globoka nevronske mreža sestavljena, je potrebno še dodatno učenje z vhodno matriko $\mathbf{X}$ in izhodno matriko $\mathbf{Y}$, vendar je običajno število potrebnih iteracij za učenje precej manjše.

Za primer iz slike 2.1 sestavimo eno ciljno nevronsko mrežo, sestavljeno iz treh enonivojskih mrež iz drugega koraka, tako da za preslikavo uporabimo uteži in pristranske vrednosti vseh prejšnjih enonivojskih mrež. Končna mreža ima zato dimenzije (3, 3, 4, 2).

Pri opisani inicializaciji uteži lahko pride do težave, kjer imamo podatke z malim številom atributov, saj lahko z matrično faktorizacijo le zmanjšamo število atributov, kar pomeni, da bo ciljna nevronska mreža sestavljena iz nivojev, kjer bo vsak naslednji nivo imel manj ali enako število nevronov. Ena izmed tehnik, da odpravimo to pomankljivost, je jedrni trik (angl. kernel trick), kjer prvotno matriko podatkov razširimo, da dobimo večje število atributov. Ta tehnika bi bila uporabna na primeru podatkov Iris, saj bi v nasprotnem primeru lahko zgradili nivoje z največ štirimi nevroni. Največkrat ta trik ni potreben, saj za podatke z le nekaj atributi ne gradimo zelo globokih nevronskih mrež.

Poglavje 5

Rezultati

V tem razdelku predstavimo rezultate na podatkovni množici MNIST in Jester jokes za problem klasifikacije in regresije. Iz dobljenih podatkov je razvidno, da je predlagana inicializacija najbolj uspešna pri začetnem nastavljanju uteži za globoke nevronske mreže in da precej pospeši hitrost učenja.

V naslednjih razdelkih niso prikazani vsi algoritmi, z vsemi različnimi parametri, saj obstaja preveliko število kombinacij parametrov, ki bi jih morali primerjati. V nekaterih razdelkih je prikazan le en algoritem in njegove izboljšave oziroma kombinacije parametrov.

Rezultati so ustvarjeni s pomočjo programskega jezika Python. Za pomoč pri izdelavi je bilo uporabljeno večje število knjižnic. Osnovna knjižnica za nevronske mreže je bila knjižnica Keras [3], ki deluje kot ovoj okrog knjižnice Tensorflow in precej poenostavi gradnjo različnih tipov modelov. Za matrično faktorizacijo je bila primarno uporabljena knjižnica scikit-learn, ki vsebuje večjo zbirko algoritmov za obdelavo matrik. Za nekatere algoritme, ki jih scikit-learn ne podpira, je bila uporabljena nekoliko zastarela knjižnica PyMF, katere specializacija so predvsem algoritmi za matrično faktorizacijo.

Ker so algoritmi za matrično faktorizacijo stohastični, lahko med posameznimi zagoni dobimo nekoliko drugačne rezultate. Rezultati se skoraj nikoli ne razlikujejo za več kot 5%. V primeru da imajo rezultati večjo varianco, pa je zraven tabel in slik dodan komentar. Vsaka črta, ki prikazuje klasifi-

kacijsko točnost oziroma MAE je povprečje treh zagonov. Prav tako so vse vrednosti v tabelah povprečje treh zagonov algoritma z enakimi parametri.

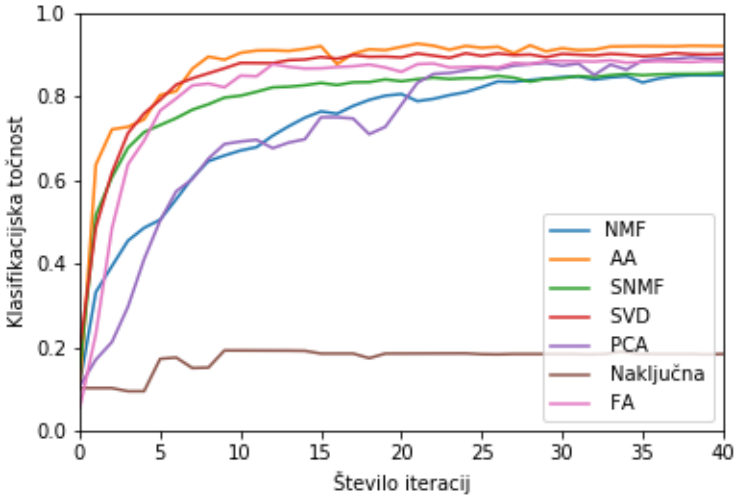
V rezultatih se pojavita tudi izraza iteracija (angl epoch) in velikost paketa (angl. batch size). Iteracija je definirana kot en prehod čez celotno množico podatkov, velikost paketa pa kot skupek učnih primerov, ki ga uporabimo, ko izračunavamo gradient. Če podatkovna množica vsebuje 100 primerov, velikost paketa pa je nastavljena na 20, potem bomo v eni iteraciji opravili 5 posodobitev matrike uteži, kjer so gradienti pri vsaki posodobitvi povprečje 20 posameznih gradientov.

5.1 Klasifikacija - MNIST

MNIST je ena izmed najbolj znanih podatkovnih zbirk za klasifikacijo, ki vsebuje slike ročno napisanih črno-belih številk med 0 in 9. Obstaja več različnih vrst podatkov MNIST, ki se razlikujejo po velikosti slik. Običajno so te slike velikosti 28×28 ali 20×20 točk. V tem primeru je uporabljena množica podatkov z 20×20 točk. Podatkovna množica, na kateri izvajamo vse nadaljnje teste, vsebuje 5000 primerov slik, ki jih klasificiramo v 10 različnih razredov. V vseh primerih je množica razdeljena na podatke za učenje (60%) in podatke za testiranje (40%).

5.1.1 Tipi matrične faktorizacije

Za izračun zgoščene predstavitve podatkov na posameznih nivojih obstaja več algoritmov. Nekaj izmed njih je opisanih v razdelku Tipi matrične faktorizacije 3.2. Slika 5.1 primerja hitrost učenja ciljne nevronske mreže glede na tip uporabljene matrične faktorizacije za globino (400, 300, 200, 100, 70, 50, 40, 30, 20, 15, 13, 10).



Slika 5.1: Primerjava vpliva različnih tipov matrične faktorizacije na klasifikacijsko točnost ciljne mreže v odvisnosti od števila iteracij učenja.

Najslabše rezultate pri klasifikaciji dosega inicializacija s pomočjo naključnih uteži, kjer se točnost giblje okrog 20%. Vse druge inicializacije največkrat dosegajo precej večjo točnost, vendar se zelo redko zgodi, da tudi te napačno inicializirajo uteži, kar privede do učenja, podobnega kot pri naključni inicializaciji.

Vse metode se v primerjavi z navadno inicializacijo odrežejo precej dobro, čeprav je med njimi nekaj odstotkov razlike. Najboljšo klasifikacijsko točnost največkrat doseže algoritem AA, najnižjo pa PCA. Vse ostale metode so običajno nekje vmes, med AA in PCA in ob vsakem novem izračunu dosežejo nekoliko različen rezultat, saj je začetna inicializacija pri matričnih faktorizacijah naključna, kar vpliva tudi na končen rezultat klasifikacije ciljne nevronske mreže.

5.1.2 Primerjava števila iteracij mreže

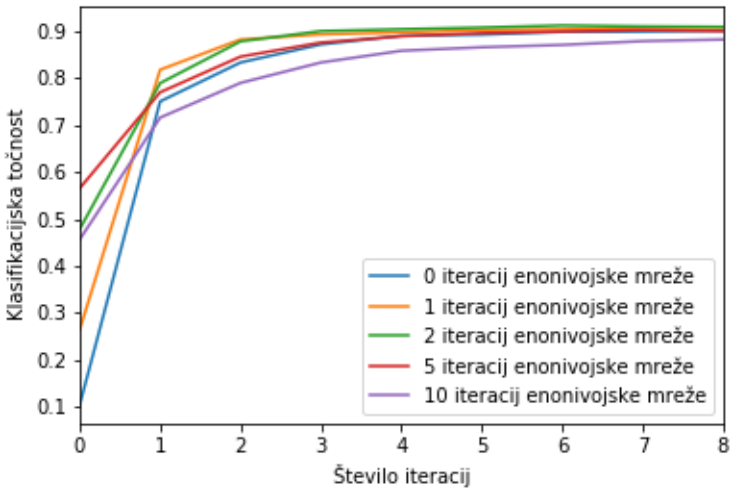
V tabeli 5.1 je podana primerjava, kako število iteracij (angl. epoch) in velikost paketa (angl. batch), na posameznih enonivojskih nevronskih mrežah

vpliva na začetno klasifikacijsko točnost ciljne nevronske mreže. Za generiranje vmesnih matrik z manjšimi dimenzijami je uporabljen algoritem nenegativne matrične faktorizacije. Tabela prikazuje klasifikacijsko točnost za mrežo z nivoji (400, 50, 30, 10), preden nevronska mreža sploh začne z učenjem.

Tabela 5.1: Primerjava klasifikacijske točnosti pred začetkom učenja ciljne nevronske mreže v odvisnosti od števila prehodov učne množice (E) in velikosti paketa (B) pri uporabi NMF inicializacije.

E \B	1	2	3	5	7	10	20
0	0.1057	0.106	0.097	0.1006	0.0982	0.0948	0.1063
1	0.3769	0.2611	0.2969	0.1595	0.1641	0.1305	0.1245
2	0.5033	0.4481	0.4049	0.3301	0.2214	0.2353	0.1446
3	0.5379	0.5188	0.4687	0.4278	0.3495	0.3075	0.215
4	0.5401	0.5309	0.5009	0.4741	0.4704	0.416	0.2516
5	0.5105	0.5436	0.53	0.4944	0.4886	0.4213	0.3359
8	0.4428	0.5033	0.5224	0.5411	0.5036	0.4898	0.3854
10	0.4303	0.4545	0.4921	0.5335	0.5283	0.5085	0.4738
14	0.4307	0.439	0.4632	0.4985	0.5107	0.5278	0.4922
20	0.4291	0.4283	0.4291	0.4302	0.4687	0.4882	0.5323

Ker so uporabljeni podatki MNIST, je pričakovana točnost brez učenja okrog 10%, kar je razvidno tudi iz prve vrstice tabele. Iz tabele sta razvidna tudi trenda, kjer med učenjem pride do prevelikega prileganja enonivojskih nevronskih mrež, kar posledično privede do prevelikega prileganja ciljne nevronske mreže, in trend, kjer večji kot je paket pri učenju, več iteracij je potrebno, preden pride do prevelikega prileganja, čeprav so končni rezultati za majhene pakete in majhno število iteracij zelo podobni rezultatom z večjim paketom in večjim številom iteracij.



Slika 5.2: Začetna klasifikacijska točnost ciljne nevronske mreže pri različnem številu iteracij enonivojskih mrež.

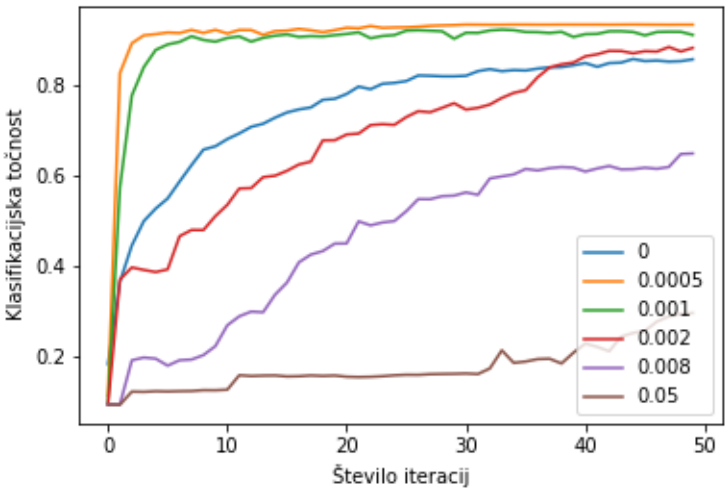
Slika 5.2 prikazuje spreminjanje začetne klasifikacijske točnosti ciljne nevronske mreže glede na število iteracij enonivojskih mrež pri konstantni velikosti paketa učenja. Iz slike je razvidno, da izbor števila iteracij v enonivojskih mrežah precej vpliva na klasifikacijsko točnost v prvi iteraciji ciljne mreže, kjer inicializacija s pomočjo matrične faktorizacije doseže boljše rezultate, vendar lahko pride do prevelikega prileganja, če enonivojske mreže učimo predolgo časa. To je razvidno iz učenja na 10 iteracijah, kjer že v začetku dosežemo slabšo točnost, kot pri učenju s 5 iteracijami.

Pri učenju z zelo velikim številom iteracij lahko tako hitro pride do prevelikega prileganja, da se inicializacija z matrično faktorizacijo obnese slabše kot naključna inicializacija.

5.1.3 Regularizacija

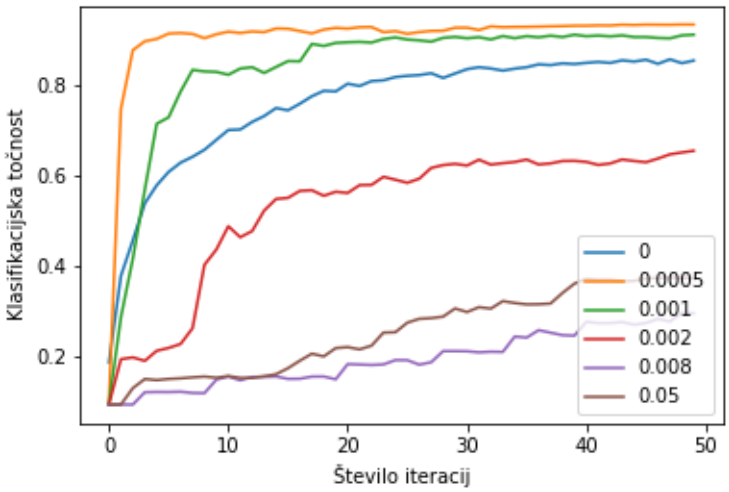
Pri učenju enonivojskih nevronskih mrež običajno hitro pride do prevelikega prileganja testnim podatkom, kar posledično privede tudi do prevelikega prileganja ciljne nevronske mreže. V tem delu so opisani rezultati vpliva regula-

rizacij enonivojskih nevronskih mrež na klasifikacijsko točnost med učenjem ciljne nevronske mreže z velikostjo nivojev (400, 300, 200, 100, 70, 50, 30, 20, 10).



Slika 5.3: Klasifikacijska točnost med učenjem ciljne globoke nevronske mreže v odvisnosti od števila iteracij učenja ciljne mreže z različnimi stopnjami regularizacije L2 enonivojskih nevronskih mrež.

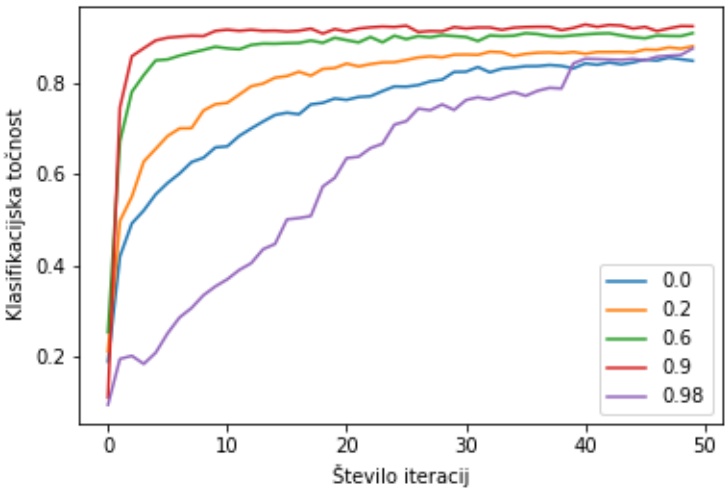
Slika 5.3 prikazuje spreminjanje klasifikacijske točnosti med učenjem ciljne nevronske mreže pri uporabi različnih faktorjev regularizacije L2. Iz podatkov je razvidno, da regularizacija enonivojskih nevronskih mrež precej vpliva na klasifikacijo ciljne nevronske mreže. Če regularizacije ne uporabimo, potem lahko na začetku pričakujemo nekoliko boljšo točnost, vendar se ta med učenjem izboljšuje precej počasneje, kot če pri učenju nastavimo majhen faktor regularizacije. V primeru, da je regularizacija na enonivojskih mrežah prevelika, se ciljna nevronska mreža uči precej počasneje, kar je razvidno pri faktorju regularizacije 0,05.



Slika 5.4: Klasifikacijska točnost med učenjem ciljne globoke nevronske mreže v odvisnosti od števila iteracij učenja ciljne mreže z različnimi stopnjami regularizacije L1 enonivojskih nevronskih mrež.

Pri uporabi regularizacije L1, prikazane na sliki 5.4, je rezultat precej podoben regularizaciji L2 in velja isto, da izbor prevelikega ali premalega faktorja regularizacije precej upočasni učenje ciljne nevronske mreže.

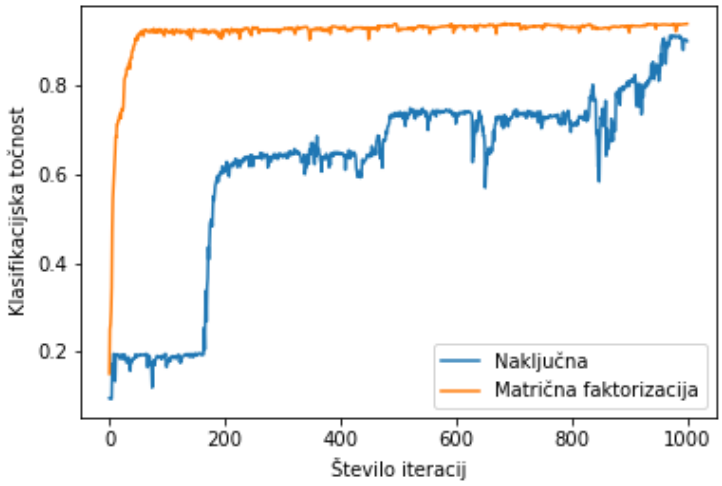
Regularizacija Dropout se od regularizacij L1 in L2 razlikuje po tem, da med delovanjem izključi določeno število nevronov, med tem ko drugi dve regularizaciji kaznujeta le velikost uteži. Slika 5.5 prikazuje odstotek nevronov, ki so bili med učenjem izključeni, in končno uspešnost napovedi. Razvidno je, da je učenje potekalo najhitrejše, ko so bili izhodi nevronov nastavljeni na nič v 90% primerov. V primeru povečanja verjetnosti izklopa nevronov pa točnost ciljne nevronske mreže hitro pade na klasifikacijsko točnost, precej slabšo od učenja brez regularizacije Dropout.



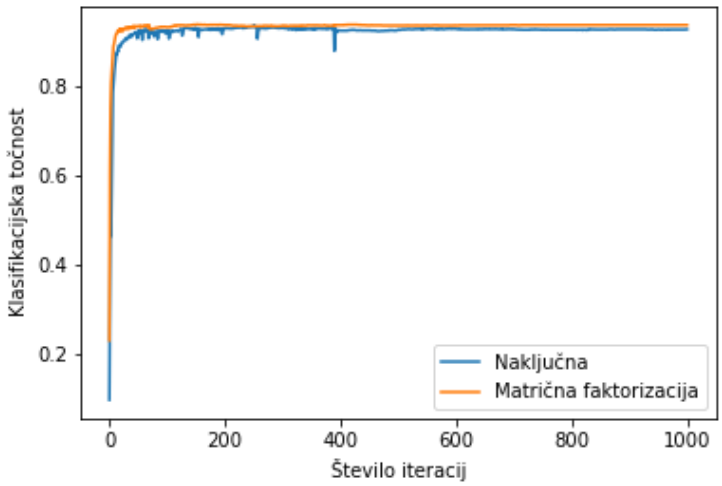
Slika 5.5: Klasifikacijska točnost med učenjem ciljne globoke nevronske mreže v odvisnosti od števila iteracij učenja ciljne mreže z različnimi stopnjami regularizacije Dropout enonivojskih nevronskih mrež.

5.1.4 Globina mreže

Največja prednost predlagane inicializacije uteži je hitrost učenja pri bolj globokih nevronskih mrežah. Slika primerja učenje globoke nevronske mreže z inicializacijo Xavier in nevronske mreže, inicializirane s pomočjo matrične faktorizacije. Z namenom je struktura mreže globoka (400, 300, 200, 100, 70, 50, 40, 30, 20, 15, 13, 10) z 10 skritimi nivoji, saj se največja prednost predlagane inicializacije pokaže šele pri hitrosti učenja globljih nevronskih mrež. Slika 5.6 prikazuje spreminjanje klasifikacijske točnosti po vsaki iteraciji. Klasifikacijska točnost pri inicializaciji z matrično faktorizacijo doseže vrh že po 40 iteracijah čez celotno učno množico, medtem ko klasična inicializacija doseže vrh šele pri 1000 iteracijah.



Slika 5.6: Klasifikacijska točnost globoke ciljne nevronske mreže z naključno in predlagano inicializacijo v odvisnosti od števila učnih iteracij ciljne nevronske mreže.

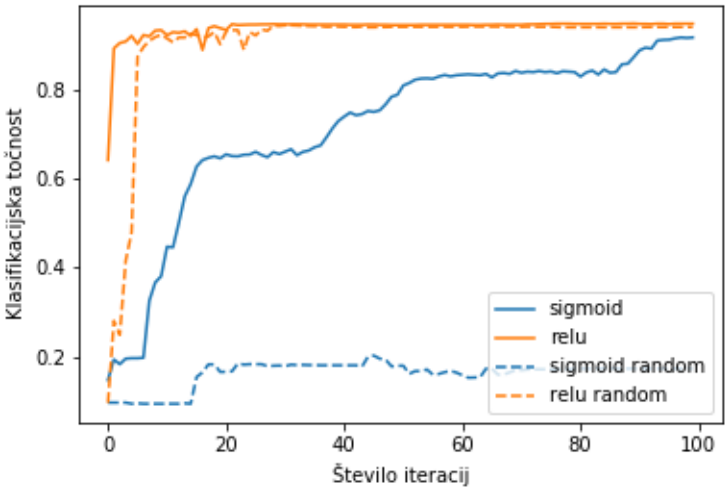


Slika 5.7: Klasifikacijska točnost plitve ciljne nevronske mreže z naključno in predlagano inicializacijo v odvisnosti od števila učnih iteracij ciljne nevronske mreže.

Pri manjših nevronskih mrežah predlagana inicializacija nima takšnega vpliva, saj med učenjem ne pride do tako velike izgube gradienta. Slika 5.7 prikazuje klasifikacijsko točnost za nevronsko mrežo s polno povezano arhitekturo (400, 200, 100, 50, 15, 10). Med inicializacijo Xavier in inicializacijo s pomočjo matrične faktorizacije ni veliko razlike. Običajno ima ciljna nevronska mreža, inicializirana z matrično faktorizacijo, nekoliko višjo klasifikacijsko točnost ob začetku učenja, vendar ta prednost izgine po nekaj iteracijah.

5.1.5 Aktivacijske funkcije in funkcije napake

Nevronske mreže lahko uporabljajo precej različnih aktivacijskih funkcij. Dve izmed najbolj pogostih sta sigmoidna in ReLu. Prednost funkcije ReLu je ta, da med učenjem ne pride do izginjanja gradienta. V tem razdelku primerjamo hitrost učenja mrež s sigmoidno funkcijo in funkcijo ReLu.



Slika 5.8: Klasifikacijska točnost pri uporabi ReLu in sigmoidne funkcije v odvisnosti od števila učnih iteracij ciljne mreže.

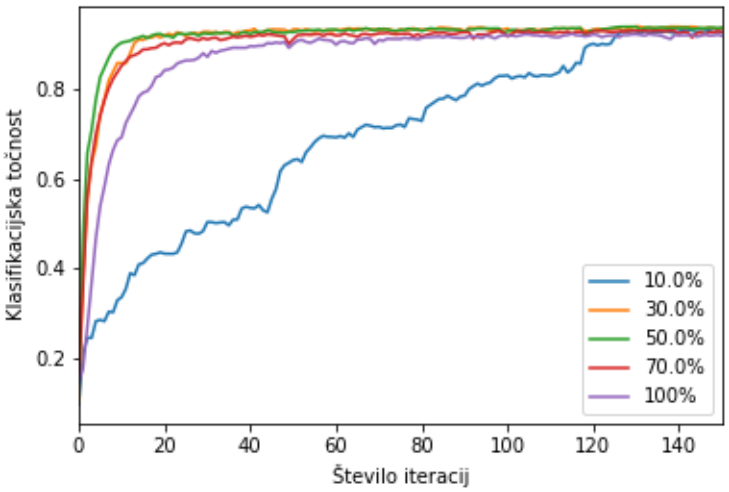
Med sigmoidno in funkcijo ReLu je pri hitrosti učenja precejšna razlika. Iz slike 5.8 je razvidno, da učenje z funkcijo ReLu poteka precej hi-

tro in da nevronska mreža že po nekaj iteracijah napoveduje s precejšnjo točnostjo. Če uporabimo inicializacijo s pomočjo matrične faktorizacije, potem hitrost učenja še nekoliko pospešimo in zvišamo klasifikacijsko točnost že pred začetkom učenja na okrog 65%. Sigmoidna funkcija za razliko od funkcije ReLU pri tej arhitekturi nevronske mreže zaradi izgube gradienta ne konvergira k utežem, ki bi dajale dobro rešitev in zato v prvih 100 iteracijah klasificira pravilno le okrog 20% cifer. Če nevronsko mrežo inicializiramo s pomočjo faktorizacije, potem se mreža zna naučiti uteži, ki dosegajo boljšo točnost, vendar je učenje vseeno počasnejše kot pri funkciji ReLu. Za generiranje je bila uporabljena struktura mreže z globino (400, 300, 200, 100, 70, 50, 40, 30, 20, 15, 13, 10).

5.1.6 Učenje matrične faktorizacije na podmnožici učne množice

Učenje s pomočjo predlagane inicializacije je običajno precej počasnejše od drugih inicializaciji, zato v tem razdelku primerjam, kakšen odstotek podatkov za učenje je potreben, da lahko zgradimo enonivojske nevronske mreže dovolj dobro, da so te še uporabne in za to faktoriziramo čim manjšo matriko podatkov in s tem pospešimo inicializacijo.

Slika 5.9 prikazuje klasifikacijsko točnost med učenjem ciljne mreže v odvisnosti od števila iteracij pri različnih odstotkih uporabljenih podatkov za matrično faktorizacijo in učenje enonivojskih mrež. Arhitektura mreže je v tem primeru (400, 300, 200, 100, 70, 50, 40, 30, 20, 15, 13, 10). Sklepamo lahko, da za podatke MNIST pri učenju inicializacije ni potrebno uporabiti vseh podatkov, vendar je dovolj že 30% podatkov. Ker so nekateri algoritmi za matrično faktorizacijo precej časovno zahtevni, kot na primer analiza arhetipov, lahko že z manjšim delom podatkov dobimo dokaj dobro aproksimacijo uteži. Če za učenje enonivojskih mrež uporabimo premalo podatkov, potem se začne klasifikacijska točnost ciljne mreže približevati naključni inicializaciji. Pri inicializaciji s celotno učno množico lahko pride tudi do prevelikega prileganja, saj se poskušajo matrike pri faktorizaciji čim bolj prilagoditi vho-



Slika 5.9: Klasifikacijska točnost med učenjem ciljne nevronske mreže v odvisnosti od števila iteracij ciljne mreže z inicializacijo, ki uporablja samo del podatkov.

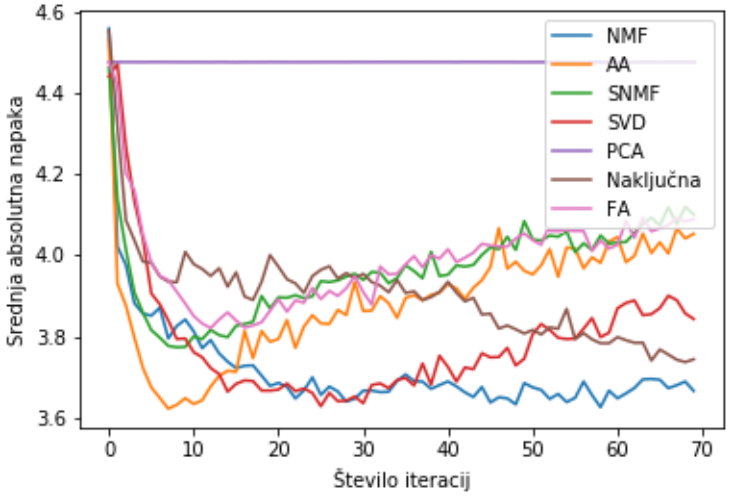
dni matriki, s tem pa se v njih pojavijo precej veliki elementi, ki kasneje slabo vplivajo na učenje enonivojskim mrež.

5.2 Regresija - Jester jokes

Za primerjavo delovanja inicializacije z matrično faktorizacijo je uporabljena podatkovna zbirka, imenovana Jester jokes, ki je sestavljena iz ocen, ki so jih posamezni uporabniki dali določenim šalam. Zbirka vsebuje ocene 7880 uporabnikov, ki so ocenili vsaj 90 od 100 šal z oceno med -10 in 10. Vse mankajoče ocene uporabnikov so bile pred začetkom učenja zapolnjene s povprečjem ocen posamezne šale. Za napovedovanje smo iz matrike ocen odstranili ocene za eno izmed šal, katera predstavlja vrednosti, ki se jih želimo z nevronsko mrežo naučiti. Za potrebe testiranja je bila množica pred učenjem razdeljena na testni (30%) in učni del (70%).

5.2.1 Tipi matrične faktorizacije

V tem razdelku primerjamo vpliv različnih tipov matrične faktorizacije na MAE ciljne nevronske mreže. Najpočasnejša a najboljša inicializacija je ponovno inicializacija s pomočjo analize arhetipov iz slike 5.10, vendar pri tej inicializaciji dokaj hitro pride do prevelikega prileganja podatkom. Pri matričnih faktorizacijah SVD in NMF so rezultati nekoliko slabši kot pri AA, vendar kljub temu dosežejo precej dobro napoved. Še nekoliko slabši so rezultati pri inicializaciji SNMF in FA, kjer ciljna nevronska mreža optimalno napoved doseže po nekaj iteracijah, vendar je ta precej slabša kot inicializacija z AA. Najslabše se je obnesla inicializacija s PCA, kjer je inicializacija tako slaba, da se ciljna mreža sploh ne nauči nobenega uporabnega znanja in vedno napoveduje z isto napako. V vseh primerih je imela arhitektura strukturo (99, 95, 90, 80, 70, 50, 40, 30, 15, 10, 5, 1).



Slika 5.10: Primerjava vpliva različnih tipov matrične faktorizacije na MAE ciljne mreže v odvisnosti od števila iteracij učenja.

5.2.2 Primerjava števila iteracij mreže

Tabela 5.2 prikazuje začetno MAE ciljne nevronske mreže pred začetkom učenja za različno število iteracij in velikosti paketov pri učenju enonivojskih mrež. Uporabljena je bil arhitektura (99, 70, 30, 1) z le dvema skritima nivojema. Razvidno je, da izbor števila iteracij in velikosti paketov za podatke šal nima vpliva na kakovost napovedi pred začetkom učenja ciljne mreže. Za isto arhitekturo mreže je naključna inicializacija brez učenja dosegala točnost MAE približno 4,525. Rezultat je precej drugačen, kot je bil pri klasifikaciji, kjer so plitve mreže, inicializirane z matrično faktorizacijo, že pred začetkom učenja dajale precej boljše rezultate (50%) kot naključna inicializacija (10%).

Tabela 5.2: Primerjava MAE pred začetkom učenja ciljne nevronske mreže v odvisnosti od števila prehodov učne množice (E) in velikosti paketa (B) na enonivojskih mrežah.

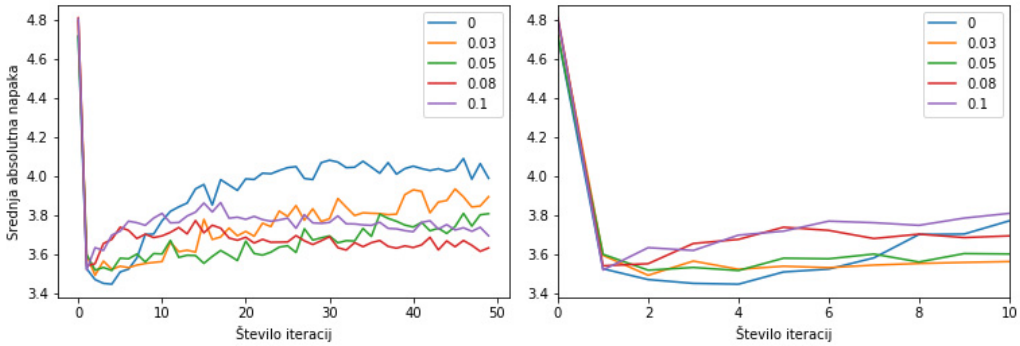
E \B	1	2	3	5	7	10	20
0	4.6262	4.6913	4.7904	4.8790	4.9238	4.5670	4.3270
1	4.8671	4.6766	6.1319	4.7182	4.6531	4.7333	4.7990
2	4.8740	4.9578	4.7450	4.0354	4.5597	5.2449	4.6163
3	4.6344	4.7066	4.7484	4.8206	4.8966	5.0648	4.8274
4	4.7492	4.7831	4.7926	4.6192	4.6689	4.7225	4.7623
5	4.8225	4.9589	5.0642	5.0057	4.4179	4.7591	4.6076
8	4.6544	4.6982	4.7375	4.7637	4.9015	4.9829	5.1595
10	4.8478	4.5601	4.5838	4.6515	4.6845	4.7141	4.7459
14	4.8195	4.9210	5.0321	5.2430	4.7184	4.5426	4.6081
20	4.6598	4.6735	4.6894	4.7665	4.8176	4.8986	5.0505

5.2.3 Regularizacija

Med učenjem nevronske mreže na podatkih šal dokaj hitro pride do prevelikega prileganja. Slika 5.11 prikazuje graf, kjer je razvidno, da že po prvih nekaj iteracijah ciljna nevronska mreža doseže optimalno točnost, ob nadalj-

njem učenju pa ta točnost začne padati oziroma pride do prevelikega prileganja. Ta efekt lahko nekoliko omilimo s pomočjo regularizacije enonivojskih mrež. Pri vseh testih regularizacije je uporabljena arhitektura (99, 80, 50, 30, 15, 5, 1).

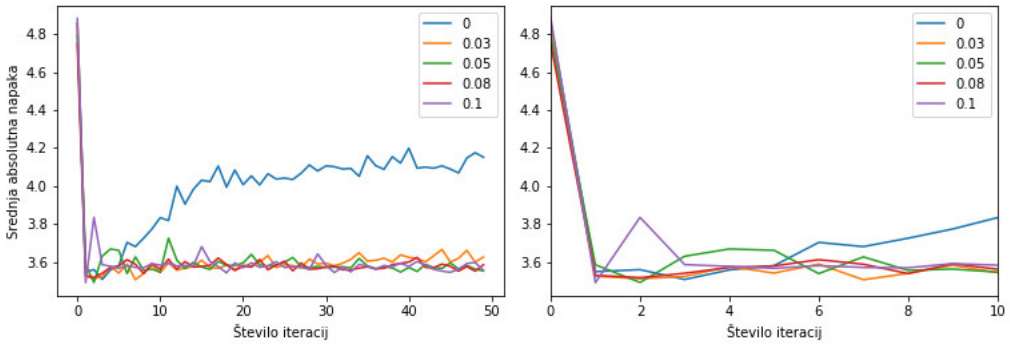
Stopnje regularizacije L2, prikazane na sliki 5.11, precej upočasnijo pojav prevelikega prileganja. Izbor stopnje regularizacije je zelo pomemben, saj ob premajhni vrednosti regularizacije točnost ciljne nevronske mreže po dosegu optimalne točnosti začne hitro padati zaradi prevelikega prileganja podatkom. To je razvidno, ko ne uporabimo nobene regularizacije.



Slika 5.11: MAE med učenjem ciljne nevronske mreže z različnimi stopnjami regularizacije L2 enonivojskih nevronskih mrež v odvisnosti od števila iteracij učenja ciljne mreže.

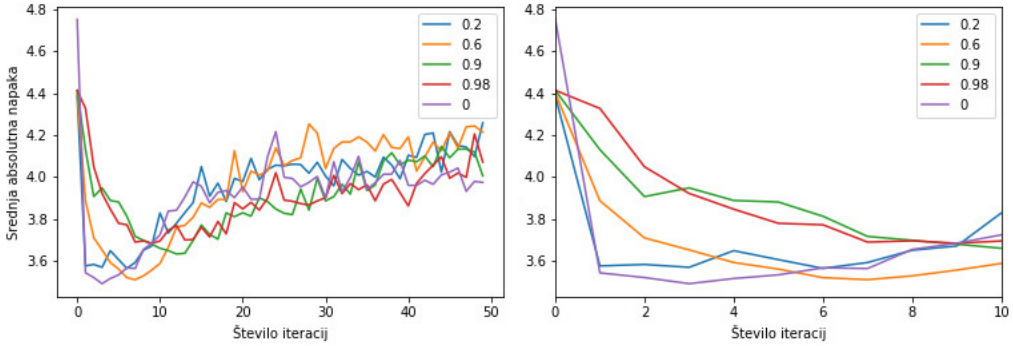
Regularizacija L1 iz slike 5.12 na točnost vpliva podobno kot regularizacija L2, le da ta regularizacija nekoliko bolj preprečuje ponovno poslabšanje točnosti oziroma preveliko prileganje napovedi ciljne mreže, ne glede na izbor faktorja regularizacije.

Regularizacija Dropout je za podatke šal delovala precej slabo. Slika 5.13 prikazuje odstotek izključenih nevronov na posameznih enonivojskih mrežah in vpliv teh na učenje ciljne nevronske mreže. Za določene faktorje regularizacije je regularizacija Dropout celo poslabšala rezultate nevronske mreže. To je vidno iz faktorjev regularizacije 0,6 in 0,9, kjer ciljna mreža dosega



Slika 5.12: MAE med učenjem ciljne nevronske mreže z različnimi stopnjami regularizacije L1 enonivojskih nevronskih mrež v odvisnosti od števila iteracij učenja ciljne mreže.

nekaj odstotkov slabšo najboljo točnost.

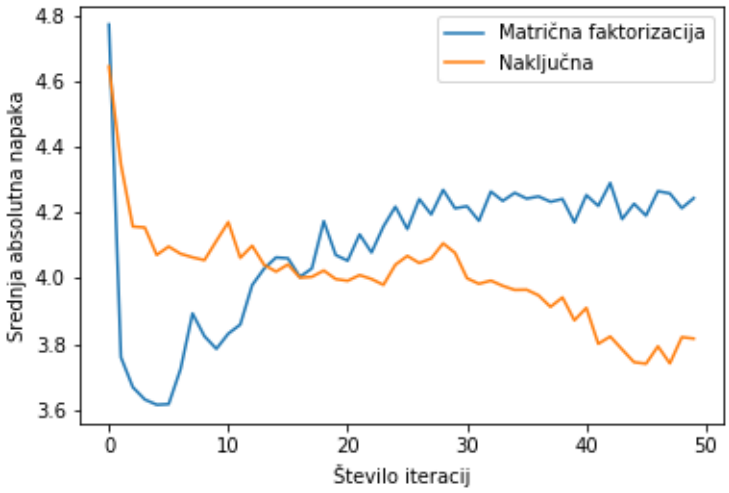


Slika 5.13: MAE med učenjem ciljne nevronske mreže z različnimi stopnjami regularizacije Dropout enonivojskih nevronskih mrež v odvisnosti od števila iteracij učenja ciljne mreže.

5.2.4 Globina mreže

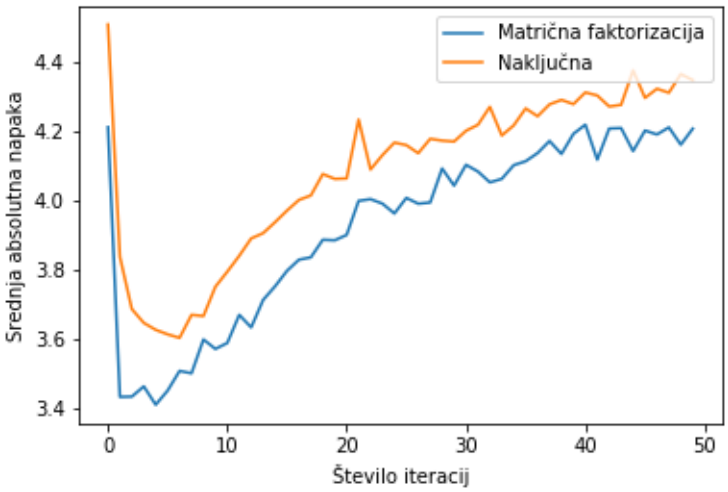
Pri regresiji lahko za napovedovanje ocen opazimo podobne rezultate, kot pri klasifikaciji ročno napisanih številk. Z inicializacijo uteži z matrično faktori-

zacijo lahko precej pospešimo učenje za globoke nevronske mreže, kot je to razvidno iz slike 5.14. Naključna inicializacija za optimalno napovedovanje na testni množici potrebuje skoraj 50 iteracij, medtem ko pri predlagani inicializaciji dosežemo najboljšo napoved že po 5 iteracijah. Pri obeh napovedih pride ob predolgem učenju precej hitro do prevelikega prileganja, zato začne napaka ponovno rasti. Pri globoki inicializaciji je bila uporabljena mreža globine (99, 95, 90, 80, 70, 60, 50, 40, 30, 15, 10, 5, 1).



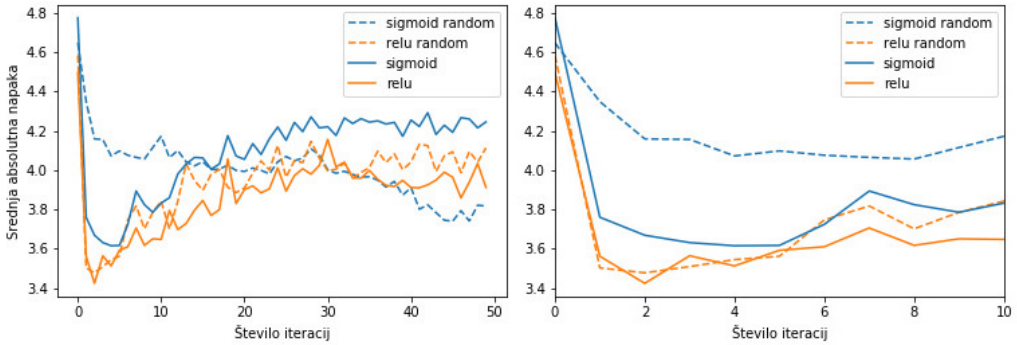
Slika 5.14: MAE ciljne globoke mreže z naključno in predlagano inicializacijo v odvisnosti od števila učnih iteracij ciljne mreže.

Pri plitvih nevronskih mrežah iz slike 5.15 je inicialiazacija nekoliko manj učinkovita, saj se hitrost učenja ne pospeši toliko kot pri globokih mrežah, vendar kljub temu običajno s to inicializacijo dosežemo nekoliko boljšo točnost kot pri naključni inicializaciji za problem napovedovanja ocen šal. Arhitektura ciljne mreže je v tem primeru (99, 50, 30, 10, 1).



Slika 5.15: MAE ciljne plitve mreže z naključno in predlagano inicializacijo v odvisnosti od števila učnih iteracij ciljne mreže.

5.2.5 Aktivacijske funkcije in funkcije napake



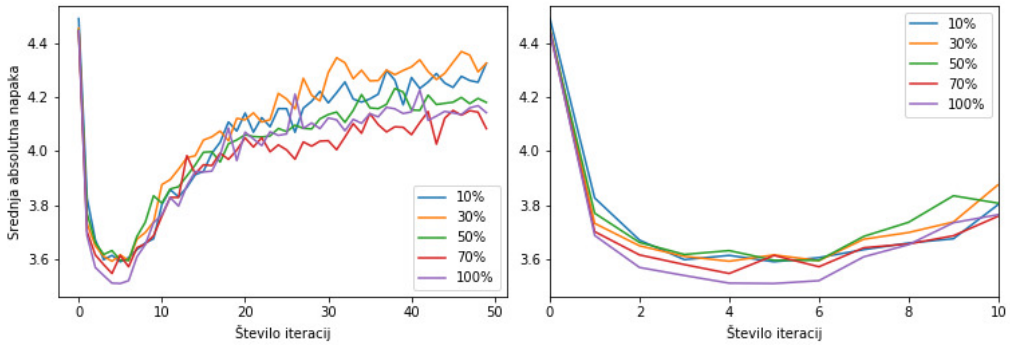
Slika 5.16: MAE pri uporabi sigmoidne funkcije in funkcije ReLu v odvisnosti od števila učnih iteracij ciljne mreže.

Pri aktivacijskih funkcijah je razlika med ReLu in sigmoidno funkcijo pri arhitekturi ciljne mreže (99, 95, 90, 80, 70, 50, 40, 30, 15, 10, 5, 1) pre-

cejšna. Kot je razvidno iz slike 5.16, je konvergenca pri funkciji ReLu precej hitrejša, napovedi pa so nekoliko boljše kot s sigmoidno funkcijo. Matrična faktorizacija na primeru teh podatkov precej pospeši hitrost pri uporabi sigmoidne funkcije, medtem ko pri funkciji ReLu med naključno in predlagano inicializacijo skoraj ni razlike.

5.2.6 Učenje matrične faktorizacije na podmnožici učne množice

Slika 5.2.6 prikazuje učenje inicializacije le na delu učne množice. Razvidno je, da ima velikost množice vpliv na MAE ciljne nevronske mreže, vendar ta vpliv ni tako velik kot pri klasifikaciji podatkov MNIST. Pri učenju ni nobenih težav predstavljal problem prevelikega prileganja, ki nastane pri matrični faktorizaciji, saj je bila najbolj uspešna prav inicializacija, ki je uporabljala celotno učno množico.



Slika 5.17: MAE ciljne nevronske mreže v odvisnosti od števila iteracij ciljne mreže z inicializacijo, ki uporablja samo del podatkov.

5.3 Čas, potreben za inicializacijo

V tem razdelku primerjamo razlike pri hitrost inicializacije za različne algoritme matrične faktorizacije in inicializacijo s pomočjo le dela učne množice,

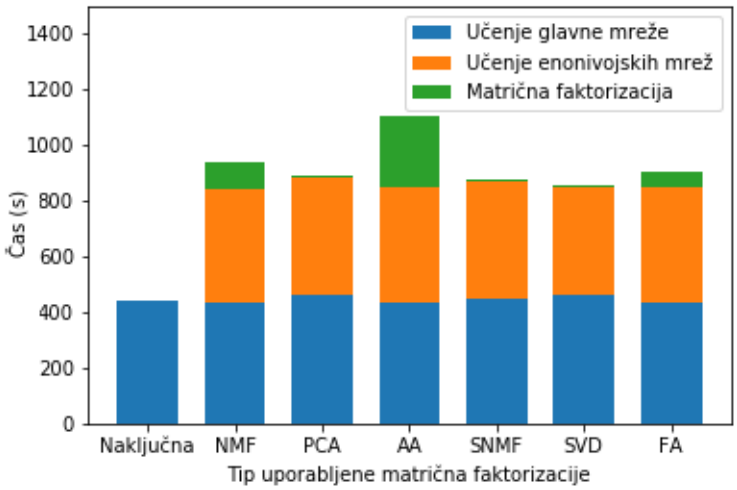
na podatkovni množici šal Jester. Pri generiranju podatkov je bila uporabljena ciljna nevronska mreža z arhitekturo (99, 95, 90, 80, 70, 50, 40, 30, 15, 10, 5, 1), velikostjo paketa 10 in število iteracij 70. Enonivojske mreže so pri učenju uporabljale velikost učnega paketa 4 in število iteracij 5, saj se je ta kombinacija izkazala za precej dobro.

5.3.1 Klasifikacijska točnost pri konstantnem številu iteracij

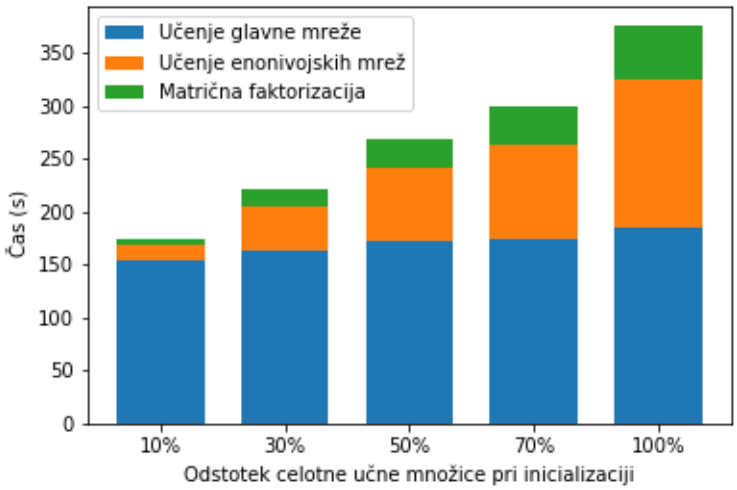
Slika 5.18 prikazuje čas inicializacije pri uporabi različnih algoritmov matrične faktorizacije za ciljno arhitekturo (99, 95, 90, 80, 70, 50, 40, 30, 15, 10, 5, 1). Čas primerjave med algoritmi ni najbolj zanesljiv, saj je za velik del razlik med tipi faktorizacije odgovorna prav implementacija posamezne faktorizacije. Implementacije, ki so del paketa scikit-learn, običajno v notranjosti uporabljajo precej optimizirano programsko kodo jezika C, medtem ko PyMF uporablja nekatere pakete za optimizacijo in množenje matrik, vendar še vseeno precej procesiranja izvaja Python, ki je nekajkrat počasnejši od jezika C.

Druga težava, ki vpliva na čas delovanja, je še rezervacija pomnilnika in prenos podatkov na grafično kartico. Ob vsaki novi mreži je zaradi implementacije Keras pomnilnik rezerviran na novo, ob tem pa se prenese še celotna množica podatkov. Posledica tega je, da se učenje prve enonivojske mreže izvede tudi do trikrat hitreje kot učenje zadnje mreže. Zaradi konsistentnosti meritev je bilo med učenjem vsakega primera okolje Python na novo zagnano, s čimer je bil problem odpravljen.

Slika 5.19 prikazuje inicializacijo nevronskih mrež, kjer za matrično faktorizacijo in učenje enonivojskih mrež uporabljamo podmnožico učnih podatkov (učenje ciljne nevronske mreže poteka na celotni učni množici). Razvidno je, da je ta inicializacija precej potratna, saj lahko pri nekaterih mrežah predstavlja večino časa učenja. Čeprav je inicializacija s celotno učno množico precej potratna, nam ta ne zagotavlja, da bo boljša kot inicializacija le z delom množice. Za podatke šal velja, da je inicilizacija, ki se je učila na ce-



Slika 5.18: Skupni čas inicializacije in učenja pri napovedovanju ocen šal, kjer so za inicializacijo uporabljeni različni algoritmi MF in celotna učna množica.



Slika 5.19: Skupni čas inicializacije in učenja pri napovedovanju ocen šal, kjer je za inicializacijo uporabljen algoritem NMF in le del učne množice.

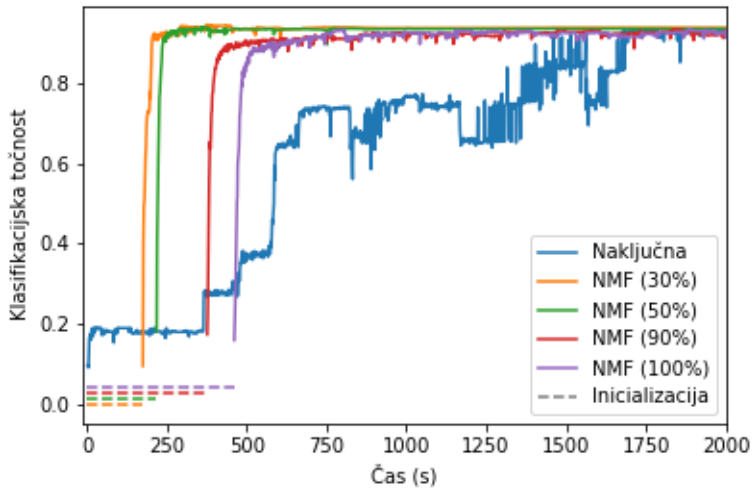
lotni učni množici, dala najboljše rezultate, kar je opisano v razdelku 5.2.6. Obratno velja za učenje na podatkih MNIST. Pri teh podatkih je najboljšo klasifikacijsko točnost dosegla inicializacija, ki je za učenje uporabila 50% učnih podatkov.

Inicializacijo bi najverjetneje lahko precej pohitrili s paralelizacijo, saj so matrične faktorizacije in učenje enonivojskih nevronskih mrež operacije, ki ne zahtevajo zaporednega izvajanja. Ta del v diplomi ni implementiran, zato se z večanjem učne množice veča tudi čas, ki je potreben za inicializacijo.

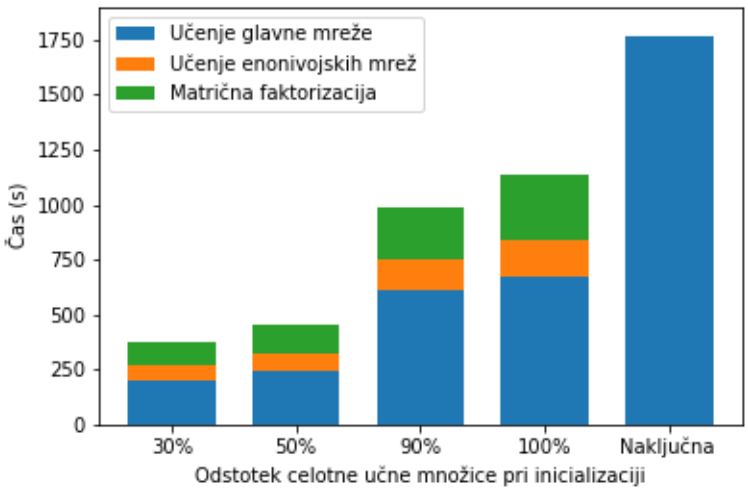
5.3.2 Klasifikacijska točnost pri variabilnem številu iteracij

Prva slika 5.20 prikazuje graf, ki opisuje spreminjanje točnosti ciljne nevronske mreže v odvisnosti od časa učenja ciljne mreže. Vse črte, ki prikazujejo učenje na podmnožici podatkov, so od začetka po osi časa zamaknjene za čas, ki je bil potreben pri inicializaciji. Ta čas prikazuje črtkana črta, kjer je vidno, da inicializacija poteka najhitreje pri uporabi 30% učne množice, najpočasnejša pa je inicializacija, kjer je uporabljena celotna množica. Predlagana inicializacija za podatke MNIST deluje zelo dobro, saj lahko že po 400 sekundah skupnega učenja dobimo klasifikacijsko točnost, za katero bi pri običajni nevronski mreži potrebovali 2000 sekund.

Bolj natančno je skupni čas učenja in inicializacije primerjan na sliki 5.21, kjer je bilo učenje ustavljeno, takoj ko se je ciljna mreža nehala učiti in klasifikacijska točnost ni več naraščala. To se pri različnih odstotkih uporabljenih podatkov zgodi ob različnih časih. Iz slike je lepo razvidno, da je lahko ta inicializacija skupaj s procesom učenja ciljne mreže pri globini mreže (400, 300, 200, 100, 70, 50, 40, 30, 20, 15, 13, 10) tudi do štirikrat hitrejša od naključne inicializacije



Slika 5.20: Primerjava skupnega časa za inicializacijo in učenje ciljne nevronske mreže v odvisnosti od klasifikacijske točnosti ciljne mreže.



Slika 5.21: Primerjava časa učenja ciljnih nevronskih mrež, kjer je za inicializacijo uporabljen algoritem NMF in le del učne množice. Učenje je bilo ustavljeno, ko klasifikacijska točnost ciljne mreže ni več naraščala.

Poglavje 6

Zaključek

V diplomski nalogi smo najprej predstavili področje nevronskih mrež in matrične faktorizacije. Opisane so tudi nekatere težave, s katerimi se nevronske mreže pogosto srečujejo in kako te težave rešujemo. Ena največjih težav še vedno ostaja časovna zahtevnost učenja mrež, zato je bila v okviru diplomske naloge predlagana nova metoda, ki za inicializacijo uteži uporablja podatke, na katerih se kasneje uči. Ta metoda je bila testirana na dveh podatkovnih množicah, na katerih se je v večini primerov izkazala za boljšo, kot naključna inicializacija uteži. Posebej dobro se je metoda izkazala za podatkovno množico MNIST na precej globokih nevronskih mrežah, kjer ob predlagani inicializaciji za enako klasifikacijsko točnost ciljne mreže s predlagano inicializacijo potrebujemo le delček iteracij, ki so potrebne pri naključni inicializaciji. Nekoliko slabše se je metoda inicializacije uteži obnesla za regresijski problem, kjer smo napovedovali ocene določenih uporabnikov, vendar je vseeno v večini primerov delovala bolje kot naključna inicializacija. V zadnjem delu je opisana še časovna zahtevnost posameznih inicializacij uteži, kjer se pokaže največja težava te inicializacije, saj je precej počasnejša od naključne inicializacije. Vendar če primerjamo čas, potreben za inicializacijo in učenje, pri različnih inicializacijah, se izkaže, da predlagana inicializacija dosega bistveno krajše čase za učenje in dosežemo tudi do štirikratno pohitritev navkljub podaljšanemu času za inicializacijo. Da bi ugotovili, kako

se predlagana inicializacija obnaša na podatkih, ki nimajo enako strukturo kot podatki MNIST ali šale Jester, bi bilo potrebno inicializacijo testirati še na več podatkovnih množicah različnih velikosti in tipov. Za pohitritev predlagane inicializacije bi lahko ustvarili še algoritem, ki bi faktorizacije in učenje enonivojskih mrež izvajal paralelno. Testirali bi lahko še verzijo, kjer bi namesto enonivojskih mrež zgradili npr. mreže s tremi nivoji. S tem bi število potrebnih faktorizacij in mrež zmanjšali za faktor tri, te mreže pa bi bile trikrat bolj globoke.

Literatura

[1] A list of cost functions used in neural networks, alongside applications. Dosegljivo: `https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications`, 2015. [Dostopano 15. 8. 2017].

[2] Sasank Chilamkurthy. Deep Learning Crash Course Part 2. Dosegljivo: `https://chsasank.github.io/deep-learning-crash-course-2.html`, 2017. [Dostopano 15. 8. 2017].

[3] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[4] Adele Cutler and Leo Breiman. Archetypal analysis. *Technometrics*, 36(4):338–347, 1994.

[5] Manuel J. A. Eugster and Friedrich Leisch. From spider-man to hero – archetypal analysis in r. 2009.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[8] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[10] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[12] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017.

[13] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[15] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[16] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[17] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[18] Rupesh Kumar Srivastava, Jonathan Masci, Faustino J. Gomez, and Jürgen Schmidhuber. Understanding locally competitive networks. *CoRR*, abs/1410.1165, 2014.

[19] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[20] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.